BOOK 17: REACT ADVANCED
Chapter 1: useReducer

WHAT IS USEREDUCER

useReducer is an alternative to useState.

Better for:
- Complex state logic
- Multiple related values
- State that depends on previous state
- Actions from many places

Similar to Redux pattern.

USEREDUCER SYNTAX

```
import { useReducer } from "react"

let [state, dispatch] = useReducer(reducer, initialState)
```

reducer - Function that updates state
initialState - Starting state value
state - Current state
dispatch - Function to trigger updates

BASIC EXAMPLE

```
function reducer(state, action) {
 switch (action.type) {
 case "INCREMENT":
 return { count: state.count + 1 }
```

```
 case "DECREMENT":
 return { count: state.count - 1 }
 case "RESET":
 return { count: 0 }
 default:
 return state
 }
}

function Counter() {
 let [state, dispatch] = useReducer(reducer, { count: 0 })

 return (
 <div>
 <p>Count: {state.count}</p>
 <button onClick={() => dispatch({ type: "INCREMENT" })}>
+ </button>
 <button onClick={() => dispatch({ type: "DECREMENT" })}>-
</button>
 <button onClick={() => dispatch({ type: "RESET" })}>Reset</
button>
 </div>
 )
}
```

THE REDUCER FUNCTION

Reducer takes current state and action.
Returns new state.

```
function reducer(state, action) {
 switch (action.type) {
 case "ACTION_TYPE":
 return newState
 default:
 return state
 }
```

```
}
```

Rules:
- Must be pure function
- No side effects
- Same input always gives same output
- Never mutate state directly

ACTIONS WITH PAYLOAD

Pass data with actions:

```
function reducer(state, action) {
 switch (action.type) {
 case "SET_NAME":
 return { ...state, name: action.payload }

 case "SET_AGE":
 return { ...state, age: action.payload }

 case "UPDATE_USER":
 return { ...state, ...action.payload }

 default:
 return state
 }
}

function UserForm() {
 let [state, dispatch] = useReducer(reducer, { name: "", age: 0 })

 return (
 <div>
 <input
 value={state.name}
 onChange={(e) => dispatch({
 type: "SET_NAME",
```

```
payload: e.target.value
})}
/>
<input
type="number"
value={state.age}
onChange={(e) => dispatch({
type: "SET_AGE",
payload: parseInt(e.target.value)
})}
/>
</div>
)
}
```

TODO LIST WITH USEREDUCER

```
function todoReducer(state, action) {
switch (action.type) {
case "ADD_TODO":
return [
...state,
{
id: Date.now(),
text: action.payload,
completed: false
}
]

case "TOGGLE_TODO":
return state.map(todo =>
todo.id === action.payload
? { ...todo, completed: !todo.completed }
: todo
)

case "DELETE_TODO":
```

```
 return state.filter(todo => todo.id !== action.payload)

 case "CLEAR_COMPLETED":
 return state.filter(todo => !todo.completed)

 default:
 return state
 }
}

function TodoApp() {
 let [todos, dispatch] = useReducer(todoReducer, [])
 let [input, setInput] = useState("")

 function handleSubmit(e) {
 e.preventDefault()
 if (!input.trim()) return
 dispatch({ type: "ADD_TODO", payload: input })
 setInput("")
 }

 return (
 <div>
 <form onSubmit={handleSubmit}>
 <input
 value={input}
 onChange={(e) => setInput(e.target.value)}
 />
 <button type="submit">Add</button>
 </form>

 <ul>
 {todos.map(todo => (
 <li key={todo.id}>
 <input
 type="checkbox"
 checked={todo.completed}
 onChange={() => dispatch({
 type: "TOGGLE_TODO",
 payload: todo.id
```

```
 })}
 />
 <span style={{
 textDecoration: todo.completed ? "line-through" : "none"
 }}>
 {todo.text}
 </span>
 <button onClick={() => dispatch({
 type: "DELETE_TODO",
 payload: todo.id
 })}>
 Delete
 </button>
 </li>
 ))}
 </ul>

 <button onClick={() => dispatch({ type: "CLEAR_COMPLETED"
})}>
 Clear Completed
 </button>
 </div>
 )
}
```

COMPLEX STATE EXAMPLE

```
function formReducer(state, action) {
 switch (action.type) {
 case "SET_FIELD":
 return {
 ...state,
 values: {
 ...state.values,
 [action.field]: action.value
 }
 }
```

```
 case "SET_ERROR":
 return {
 ...state,
 errors: {
 ...state.errors,
 [action.field]: action.error
 }
 }

 case "SET_TOUCHED":
 return {
 ...state,
 touched: {
 ...state.touched,
 [action.field]: true
 }
 }

 case "SET_SUBMITTING":
 return { ...state, isSubmitting: action.payload }

 case "RESET":
 return action.payload

 default:
 return state
 }
}

let initialState = {
 values: { email: "", password: "" },
 errors: {},
 touched: {},
 isSubmitting: false
}

function LoginForm() {
 let [state, dispatch] = useReducer(formReducer, initialState)
```

```
function handleChange(e) {
let { name, value } = e.target
dispatch({ type: "SET_FIELD", field: name, value })
}

function handleBlur(e) {
let { name } = e.target
dispatch({ type: "SET_TOUCHED", field: name })
validateField(name)
}

function validateField(field) {
let value = state.values[field]
let error = ""

if (field === "email" && !value.includes("@")) {
error = "Invalid email"
}

if (field === "password" && value.length < 6) {
error = "Password too short"
}

dispatch({ type: "SET_ERROR", field, error })
}

async function handleSubmit(e) {
e.preventDefault()
dispatch({ type: "SET_SUBMITTING", payload: true })

try {
await submitForm(state.values)
dispatch({ type: "RESET", payload: initialState })
} catch (err) {
dispatch({ type: "SET_ERROR", field: "form", error: err.message })
} finally {
dispatch({ type: "SET_SUBMITTING", payload: false })
}
}
```

```
 return (
 <form onSubmit={handleSubmit}>
 <input
 name="email"
 value={state.values.email}
 onChange={handleChange}
 onBlur={handleBlur}
 />
 {state.touched.email && state.errors.email && (
 <span className="error">{state.errors.email}</span>
 )}

 <input
 name="password"
 type="password"
 value={state.values.password}
 onChange={handleChange}
 onBlur={handleBlur}
 />
 {state.touched.password && state.errors.password && (
 <span className="error">{state.errors.password}</span>
 )}

 <button type="submit" disabled={state.isSubmitting}>
 {state.isSubmitting ? "Submitting..." : "Submit"}
 </button>
 </form>
 )
}
```

LAZY INITIALIZATION

For expensive initial state:

```
function init(initialCount) {
 return { count: initialCount }
}
```

```
function Counter({ initialCount }) {
 let [state, dispatch] = useReducer(reducer, initialCount, init)

 return (...)
}
```

init function runs only once.

USESTATE VS USEREDUCER

Use useState when:
- Simple values
- Independent state pieces
- Few update logic variations

Use useReducer when:
- Complex objects/arrays
- Related state pieces
- Many update types
- State depends on previous state
- Updates from deep components

ACTION CREATORS

Helper functions for creating actions:

```
let actions = {
 addTodo: (text) => ({ type: "ADD_TODO", payload: text }),
 toggleTodo: (id) => ({ type: "TOGGLE_TODO", payload: id }),
 deleteTodo: (id) => ({ type: "DELETE_TODO", payload: id })
}

function TodoApp() {
```

```
let [todos, dispatch] = useReducer(todoReducer, [])

return (
<ul>
{todos.map(todo => (
<li key={todo.id}>
<span>{todo.text}</span>
<button onClick={() => dispatch(actions.toggleTodo(todo.id))}
>
Toggle
</button>
<button onClick={() => dispatch(actions.deleteTodo(todo.id))}
>
Delete
</button>
</li>
))}
</ul>
)
}
```

PRACTICE

Convert useState to useReducer.

Build a todo app with useReducer.

Create form state with useReducer.

Use action creators.

Practice lazy initialization.

WHAT YOU LEARNED

You learned useReducer syntax.
You learned reducer functions.
You learned actions with payload.
You learned when to use useReducer.
You learned action creators.

The next chapter teaches advanced context patterns.
BOOK 17: REACT ADVANCED
Chapter 2: Advanced Context Patterns

CONTEXT WITH REDUCER

Combine context and reducer for global state:

```
import { createContext, useContext, useReducer } from "react"

let StateContext = createContext(null)
let DispatchContext = createContext(null)

function appReducer(state, action) {
 switch (action.type) {
 case "SET_USER":
 return { ...state, user: action.payload }
 case "SET_THEME":
 return { ...state, theme: action.payload }
 case "ADD_NOTIFICATION":
 return {
 ...state,
 notifications: [...state.notifications, action.payload]
 }
 case "REMOVE_NOTIFICATION":
 return {
 ...state,
 notifications: state.notifications.filter(
 n => n.id !== action.payload
```

```
 )
 }
 default:
 return state
 }
}

let initialState = {
 user: null,
 theme: "light",
 notifications: []
}

function AppProvider({ children }) {
 let [state, dispatch] = useReducer(appReducer, initialState)

 return (
 <StateContext.Provider value={state}>
 <DispatchContext.Provider value={dispatch}>
 {children}
 </DispatchContext.Provider>
 </StateContext.Provider>
 )
}

function useAppState() {
 let context = useContext(StateContext)
 if (context === undefined) {
 throw new Error("useAppState must be used within AppProvider")
 }
 return context
}

function useAppDispatch() {
 let context = useContext(DispatchContext)
 if (context === undefined) {
 throw new Error("useAppDispatch must be used within
AppProvider")
 }
 return context
```

```
}
```

WHY SPLIT STATE AND DISPATCH

```
function UserDisplay() {
 let { user } = useAppState()
 return <p>Hello, {user?.name}</p>
}

function LoginButton() {
 let dispatch = useAppDispatch()
 return (
 <button onClick={() => dispatch({ type: "SET_USER", payload: {
name: "Alice" } })}>
 Login
 </button>
 )
}
```

LoginButton only uses dispatch.
It won't re-render when user changes.

TYPED ACTIONS

Create action creators:

```
let actions = {
 setUser: (user) => ({ type: "SET_USER", payload: user }),
 setTheme: (theme) => ({ type: "SET_THEME", payload: theme }),
 addNotification: (notification) => ({
 type: "ADD_NOTIFICATION",
 payload: { id: Date.now(), ...notification }
 }),
 removeNotification: (id) => ({ type: "REMOVE_NOTIFICATION",
```

payload: id })
}

Create hooks with actions:

function useNotifications() {
 let { notifications } = useAppState()
 let dispatch = useAppDispatch()

 return {
 notifications,
 addNotification: (message, type = "info") => {
 dispatch(actions.addNotification({ message, type }))
 },
 removeNotification: (id) => {
 dispatch(actions.removeNotification(id))
 }
 }
}

MULTIPLE CONTEXTS

Separate concerns into different contexts:

let AuthContext = createContext(null)
let ThemeContext = createContext(null)
let CartContext = createContext(null)

function AuthProvider({ children }) {
 let [user, setUser] = useState(null)

 let value = useMemo(() => ({
 user,
 login: (userData) => setUser(userData),
 logout: () => setUser(null)
 }), [user])

```
 return (
 <AuthContext.Provider value={value}>
 {children}
 </AuthContext.Provider>
 )
}

function ThemeProvider({ children }) {
 let [theme, setTheme] = useState("light")

 let value = useMemo(() => ({
 theme,
 toggleTheme: () => setTheme(t => t === "light" ? "dark" :
"light")
 }), [theme])

 return (
 <ThemeContext.Provider value={value}>
 {children}
 </ThemeContext.Provider>
 )
}

function App() {
 return (
 <AuthProvider>
 <ThemeProvider>
 <CartProvider>
 <Layout />
 </CartProvider>
 </ThemeProvider>
 </AuthProvider>
 )
}
```

COMPOSE PROVIDERS

Avoid deep nesting:

```
function ComposeProviders({ providers, children }) {
 return providers.reduceRight(
 (acc, Provider) => <Provider>{acc}</Provider>,
 children
 )
}

function App() {
 return (
 <ComposeProviders providers={[AuthProvider, ThemeProvider,
CartProvider]}>
 <Layout />
 </ComposeProviders>
 )
}
```

CONTEXT SELECTORS

Prevent unnecessary re-renders:

```
import { useSyncExternalStore, useCallback } from "react"

function createStore(initialState) {
 let state = initialState
 let listeners = new Set()

 return {
 getState: () => state,
 setState: (newState) => {
 state = typeof newState === "function" ? newState(state) :
newState
 listeners.forEach(listener => listener())
 },
 subscribe: (listener) => {
```

```
 listeners.add(listener)
 return () => listeners.delete(listener)
 }
 }
}

let StoreContext = createContext(null)

function StoreProvider({ children }) {
 let store = useMemo(() => createStore({
 user: null,
 posts: [],
 theme: "light"
 }), [])

 return (
 <StoreContext.Provider value={store}>
 {children}
 </StoreContext.Provider>
 )
}

function useStore(selector) {
 let store = useContext(StoreContext)

 return useSyncExternalStore(
 store.subscribe,
 useCallback(() => selector(store.getState()), [store, selector])
 )
}

function useUser() {
 return useStore(state => state.user)
}

function useTheme() {
 return useStore(state => state.theme)
}
```

ASYNC ACTIONS IN CONTEXT

Handle async operations:

```
function AuthProvider({ children }) {
 let [state, dispatch] = useReducer(authReducer, {
 user: null,
 loading: false,
 error: null
 })

 async function login(credentials) {
 dispatch({ type: "LOGIN_START" })

 try {
 let response = await fetch("/api/login", {
 method: "POST",
 headers: { "Content-Type": "application/json" },
 body: JSON.stringify(credentials)
 })

 if (!response.ok) throw new Error("Login failed")

 let user = await response.json()
 dispatch({ type: "LOGIN_SUCCESS", payload: user })
 } catch (error) {
 dispatch({ type: "LOGIN_ERROR", payload: error.message })
 }
 }

 async function logout() {
 await fetch("/api/logout", { method: "POST" })
 dispatch({ type: "LOGOUT" })
 }

 let value = useMemo(() => ({
 ...state,
 login,
```

```
 logout
 }), [state])

 return (
 <AuthContext.Provider value={value}>
 {children}
 </AuthContext.Provider>
 )
}

function authReducer(state, action) {
 switch (action.type) {
 case "LOGIN_START":
 return { ...state, loading: true, error: null }
 case "LOGIN_SUCCESS":
 return { ...state, loading: false, user: action.payload }
 case "LOGIN_ERROR":
 return { ...state, loading: false, error: action.payload }
 case "LOGOUT":
 return { ...state, user: null }
 default:
 return state
 }
}
```

PERSISTED CONTEXT

Save to localStorage:

```
function createPersistedContext(key, initialState) {
 let Context = createContext(null)

 function Provider({ children }) {
 let [state, setState] = useState(() => {
 let saved = localStorage.getItem(key)
 return saved ? JSON.parse(saved) : initialState
 })
```

```
 useEffect(() => {
 localStorage.setItem(key, JSON.stringify(state))
 }, [state])

 return (
 <Context.Provider value={[state, setState]}>
 {children}
 </Context.Provider>
 )
 }

 function usePersistedState() {
 let context = useContext(Context)
 if (!context) {
 throw new Error("usePersistedState must be used within Provider")
 }
 return context
 }

 return { Provider, usePersistedState }
}

let { Provider: SettingsProvider, usePersistedState: useSettings } =
 createPersistedContext("settings", { theme: "light", language: "en" })
```

CONTEXT FACTORY

Create contexts consistently:

```
function createStateContext(useValue) {
 let StateContext = createContext(null)

 function Provider({ children }) {
 let value = useValue()

 return (
```

```
 <StateContext.Provider value={value}>
 {children}
 </StateContext.Provider>
 )
 }

 function useStateContext() {
 let context = useContext(StateContext)
 if (context === null) {
 throw new Error("useStateContext must be used within Provider")
 }
 return context
 }

 return [Provider, useStateContext]
}

let [CounterProvider, useCounter] = createStateContext(() => {
 let [count, setCount] = useState(0)

 return {
 count,
 increment: () => setCount(c => c + 1),
 decrement: () => setCount(c => c - 1)
 }
})
```

PRACTICE

Combine context with reducer.

Split state and dispatch contexts.

Create multiple contexts.

Handle async operations.

Build persisted context.

WHAT YOU LEARNED

You learned context with reducer.
You learned to split state and dispatch.
You learned multiple contexts.
You learned async actions.
You learned persisted context.
You learned context factories.

The next chapter teaches compound components.

BOOK 17: REACT ADVANCED

Chapter 3: Compound Components

WHAT ARE COMPOUND COMPONENTS

Compound components work together as a group.

Like HTML select and option:

```
<select>
 <option>One</option>
 <option>Two</option>
</select>
```

Components share implicit state.
Flexible composition.

BASIC COMPOUND COMPONENT

Toggle example:

```
import { createContext, useContext, useState } from "react"

let ToggleContext = createContext(null)

function Toggle({ children }) {
 let [on, setOn] = useState(false)

 function toggle() {
 setOn(prev => !prev)
 }

 return (
 <ToggleContext.Provider value={{ on, toggle }}>
 {children}
 </ToggleContext.Provider>
 )
}

function useToggle() {
 let context = useContext(ToggleContext)
 if (!context) {
 throw new Error("useToggle must be used within Toggle")
 }
 return context
}

function ToggleOn({ children }) {
 let { on } = useToggle()
 return on ? children : null
}

function ToggleOff({ children }) {
 let { on } = useToggle()
 return on ? null : children
}

function ToggleButton() {
```

```
 let { on, toggle } = useToggle()
 return <button onClick={toggle}>{on ? "ON" : "OFF"}</
button>
}

Toggle.On = ToggleOn
Toggle.Off = ToggleOff
Toggle.Button = ToggleButton
```

Usage:

```
function App() {
 return (
 <Toggle>
 <Toggle.On>The toggle is on!</Toggle.On>
 <Toggle.Off>The toggle is off!</Toggle.Off>
 <Toggle.Button />
 </Toggle>
 )
}
```

TABS COMPONENT

```
let TabsContext = createContext(null)

function Tabs({ children, defaultIndex = 0 }) {
 let [activeIndex, setActiveIndex] = useState(defaultIndex)

 return (
 <TabsContext.Provider value={{ activeIndex, setActiveIndex }}>
 <div className="tabs">{children}</div>
 </TabsContext.Provider>
 )
}

function useTabs() {
```

```
 let context = useContext(TabsContext)
 if (!context) {
 throw new Error("useTabs must be used within Tabs")
 }
 return context
}

function TabList({ children }) {
 return <div className="tab-list">{children}</div>
}

function Tab({ children, index }) {
 let { activeIndex, setActiveIndex } = useTabs()
 let isActive = activeIndex === index

 return (
 <button
 className={`tab ${isActive ? "active" : ""}`}
 onClick={() => setActiveIndex(index)}
 >
 {children}
 </button>
 )
}

function TabPanels({ children }) {
 return <div className="tab-panels">{children}</div>
}

function TabPanel({ children, index }) {
 let { activeIndex } = useTabs()
 return activeIndex === index ? <div className="tab-
panel">{children}</div> : null
}

Tabs.List = TabList
Tabs.Tab = Tab
Tabs.Panels = TabPanels
Tabs.Panel = TabPanel
```

Usage:

```
function App() {
 return (
 <Tabs>
 <Tabs.List>
 <Tabs.Tab index={0}>First</Tabs.Tab>
 <Tabs.Tab index={1}>Second</Tabs.Tab>
 <Tabs.Tab index={2}>Third</Tabs.Tab>
 </Tabs.List>

 <Tabs.Panels>
 <Tabs.Panel index={0}>First panel content</Tabs.Panel>
 <Tabs.Panel index={1}>Second panel content</Tabs.Panel>
 <Tabs.Panel index={2}>Third panel content</Tabs.Panel>
 </Tabs.Panels>
 </Tabs>
 )
}
```

ACCORDION COMPONENT

```
let AccordionContext = createContext(null)
let AccordionItemContext = createContext(null)

function Accordion({ children, allowMultiple = false }) {
 let [openItems, setOpenItems] = useState([])

 function toggleItem(index) {
 if (allowMultiple) {
 setOpenItems(prev =>
 prev.includes(index)
 ? prev.filter(i => i !== index)
 : [...prev, index]
 )
 } else {
```

```
 setOpenItems(prev => 
 prev.includes(index) ? [] : [index]
 )
 }
 }

 function isOpen(index) {
 return openItems.includes(index)
 }

 return (
 <AccordionContext.Provider value={{ toggleItem, isOpen }}>
 <div className="accordion">{children}</div>
 </AccordionContext.Provider>
 )
}

function useAccordion() {
 let context = useContext(AccordionContext)
 if (!context) {
 throw new Error("useAccordion must be used within Accordion")
 }
 return context
}

function AccordionItem({ children, index }) {
 return (
 <AccordionItemContext.Provider value={{ index }}>
 <div className="accordion-item">{children}</div>
 </AccordionItemContext.Provider>
 )
}

function useAccordionItem() {
 let context = useContext(AccordionItemContext)
 if (!context) {
 throw new Error("useAccordionItem must be used within
AccordionItem")
 }
 return context
```

```
}

function AccordionButton({ children }) {
 let { toggleItem, isOpen } = useAccordion()
 let { index } = useAccordionItem()

 return (
 <button
 className={`accordion-button ${isOpen(index) ? "open" : ""}`}
 onClick={() => toggleItem(index)}
 >
 {children}
 <span>{isOpen(index) ? "-" : "+"}</span>
 </button>
 )
}

function AccordionPanel({ children }) {
 let { isOpen } = useAccordion()
 let { index } = useAccordionItem()

 if (!isOpen(index)) return null

 return <div className="accordion-panel">{children}</div>
}

Accordion.Item = AccordionItem
Accordion.Button = AccordionButton
Accordion.Panel = AccordionPanel
```

Usage:

```
function FAQ() {
 return (
 <Accordion>
 <Accordion.Item index={0}>
 <Accordion.Button>What is React?</Accordion.Button>
 <Accordion.Panel>
 React is a JavaScript library for building user interfaces.
```

```
 </Accordion.Panel>
 </Accordion.Item>

 <Accordion.Item index={1}>
 <Accordion.Button>What are hooks?</Accordion.Button>
 <Accordion.Panel>
 Hooks let you use state and other React features in function
components.
 </Accordion.Panel>
 </Accordion.Item>
 </Accordion>
 )
}
```

MENU COMPONENT

```
let MenuContext = createContext(null)

function Menu({ children }) {
 let [isOpen, setIsOpen] = useState(false)
 let menuRef = useRef(null)

 useEffect(() => {
 function handleClickOutside(event) {
 if (menuRef.current && !menuRef.current.contains(event.target)) {
 setIsOpen(false)
 }
 }

 document.addEventListener("mousedown", handleClickOutside)
 return () => document.removeEventListener("mousedown",
handleClickOutside)
 }, [])

 return (
 <MenuContext.Provider value={{ isOpen, setIsOpen }}>
 <div className="menu" ref={menuRef}>
```

```
 {children}
 </div>
 </MenuContext.Provider>
 )
}

function useMenu() {
 let context = useContext(MenuContext)
 if (!context) {
 throw new Error("useMenu must be used within Menu")
 }
 return context
}

function MenuButton({ children }) {
 let { isOpen, setIsOpen } = useMenu()

 return (
 <button
 className="menu-button"
 onClick={() => setIsOpen(!isOpen)}
 >
 {children}
 </button>
 )
}

function MenuList({ children }) {
 let { isOpen } = useMenu()

 if (!isOpen) return null

 return <ul className="menu-list">{children}</ul>
}

function MenuItem({ children, onClick }) {
 let { setIsOpen } = useMenu()

 function handleClick() {
 onClick?.()
```

```
 setIsOpen(false)
 }

 return (
 <li className="menu-item" onClick={handleClick}>
 {children}
 </li>
 )
}

Menu.Button = MenuButton
Menu.List = MenuList
Menu.Item = MenuItem
```

Usage:

```
function App() {
 return (
 <Menu>
 <Menu.Button>Options</Menu.Button>
 <Menu.List>
 <Menu.Item onClick={() => console.log("Edit")}>Edit</
Menu.Item>
 <Menu.Item onClick={() => console.log("Delete")}>Delete</
Menu.Item>
 <Menu.Item onClick={() => console.log("Share")}>Share</
Menu.Item>
 </Menu.List>
 </Menu>
 )
}
```

BENEFITS OF COMPOUND COMPONENTS

Flexibility:
User controls structure.

Components work in any order.

Implicit state:
No prop drilling.
State shared through context.

Clean API:
Related components grouped.
Easy to understand usage.

Extensibility:
Add new child components easily.
Modify behavior without breaking API.

PRACTICE

Build a Toggle compound component.

Create Tabs with panels.

Build an Accordion.

Create a Menu dropdown.

Add customization options.

WHAT YOU LEARNED

You learned compound component pattern.
You learned to share state with context.
You learned to build Tabs, Accordion, Menu.
You learned benefits of compound components.

The next chapter teaches render props pattern.
BOOK 17: REACT ADVANCED
Chapter 4: Render Props Pattern

WHAT ARE RENDER PROPS

A render prop is a function prop that returns elements.

Component calls the function to determine what to render.
Allows sharing logic between components.

```
function DataProvider({ render }) {
 let [data, setData] = useState(null)

 useEffect(() => {
 fetchData().then(setData)
 }, [])

 return render(data)
}

<DataProvider render={(data) => <Display data={data} />} />
```

BASIC RENDER PROP

Mouse position tracker:

```
function MouseTracker({ render }) {
 let [position, setPosition] = useState({ x: 0, y: 0 })

 useEffect(() => {
 function handleMouseMove(event) {
 setPosition({ x: event.clientX, y: event.clientY })
```

```
 }

 window.addEventListener("mousemove", handleMouseMove)
 return () => window.removeEventListener("mousemove",
handleMouseMove)
 }, [])

 return render(position)
}

function App() {
 return (
 <MouseTracker
 render={({ x, y }) => (
 <div>
 Mouse position: {x}, {y}
 </div>
 )}
 />
 )
}
```

CHILDREN AS FUNCTION

Alternative syntax using children:

```
function MouseTracker({ children }) {
 let [position, setPosition] = useState({ x: 0, y: 0 })

 useEffect(() => {
 function handleMouseMove(event) {
 setPosition({ x: event.clientX, y: event.clientY })
 }

 window.addEventListener("mousemove", handleMouseMove)
 return () => window.removeEventListener("mousemove",
handleMouseMove)
```

```
 }, [])

 return children(position)
}

function App() {
 return (
 <MouseTracker>
 {({ x, y }) => (
 <div>
 Mouse position: {x}, {y}
 </div>
 )}
 </MouseTracker>
 )
}
```

DATA FETCHING RENDER PROP

```
function Fetch({ url, children }) {
 let [state, setState] = useState({
 data: null,
 loading: true,
 error: null
 })

 useEffect(() => {
 let cancelled = false

 async function fetchData() {
 setState({ data: null, loading: true, error: null })

 try {
 let response = await fetch(url)

 if (!response.ok) {
 throw new Error("Failed to fetch")
```

```
  }

  let data = await response.json()

  if (!cancelled) {
  setState({ data, loading: false, error: null })
  }
  } catch (error) {
  if (!cancelled) {
  setState({ data: null, loading: false, error: error.message })
  }
  }
  }

  fetchData()

  return () => {
  cancelled = true
  }
  }, [url])

  return children(state)
}

function App() {
  return (
  <Fetch url="/api/users">
  {({ data, loading, error }) => {
  if (loading) return <p>Loading...</p>
  if (error) return <p>Error: {error}</p>

  return (
  <ul>
  {data.map(user => (
  <li key={user.id}>{user.name}</li>
  ))}
  </ul>
  )
  }}
  </Fetch>
```

```
 )
}
```

TOGGLE RENDER PROP

```
function Toggle({ children, initialOn = false }) {
 let [on, setOn] = useState(initialOn)

 function toggle() {
 setOn(prev => !prev)
 }

 function setOff() {
 setOn(false)
 }

 function setOn() {
 setOn(true)
 }

 return children({ on, toggle, setOff, setOn })
}

function App() {
 return (
 <Toggle>
 {({ on, toggle }) => (
 <div>
 <button onClick={toggle}>
 {on ? "Hide" : "Show"}
 </button>
 {on && <div className="content">Hidden content!</div>}
 </div>
 )}
 </Toggle>
 )
}
```

FORM STATE RENDER PROP

```
function Form({ initialValues, children, onSubmit }) {
 let [values, setValues] = useState(initialValues)
 let [errors, setErrors] = useState({})
 let [isSubmitting, setIsSubmitting] = useState(false)

 function handleChange(name) {
 return (event) = > {
 setValues({ ...values, [name]: event.target.value })
 }
 }

 function handleSubmit(event) {
 event.preventDefault()
 setIsSubmitting(true)

 onSubmit(values)
 .then(() = > setIsSubmitting(false))
 .catch((err) = > {
 setErrors({ submit: err.message })
 setIsSubmitting(false)
 })
 }

 function reset() {
 setValues(initialValues)
 setErrors({})
 }

 return children({
 values,
 errors,
 isSubmitting,
 handleChange,
 handleSubmit,
```

```
 reset
 })
}

function LoginForm() {
 return (
 <Form
 initialValues={{ email: "", password: "" }}
 onSubmit={(values) => login(values)}
 >
 {({ values, handleChange, handleSubmit, isSubmitting }) => (
 <form onSubmit={handleSubmit}>
 <input
 type="email"
 value={values.email}
 onChange={handleChange("email")}
 placeholder="Email"
 />
 <input
 type="password"
 value={values.password}
 onChange={handleChange("password")}
 placeholder="Password"
 />
 <button type="submit" disabled={isSubmitting}>
 {isSubmitting ? "Logging in..." : "Login"}
 </button>
 </form>
 )}
 </Form>
 )
}
```

LIST RENDER PROP

```
function List({ items, children, emptyMessage = "No items" }) {
 if (items.length === 0) {
```

```
 return <p>{emptyMessage}</p>
 }

 return (
 <ul>
 {items.map((item, index) => (
 <li key={item.id || index}>
 {children(item, index)}
 </li>
 ))}
 </ul>
 )
}

function App() {
 let users = [
 { id: 1, name: "Alice", email: "alice@example.com" },
 { id: 2, name: "Bob", email: "bob@example.com" }
 ]

 return (
 <List items={users} emptyMessage="No users found">
 {(user) => (
 <div>
 <strong>{user.name}</strong>
 <span>{user.email}</span>
 </div>
 )}
 </List>
 )
}
```

CONTROLLED INPUT RENDER PROP

```
function ControlledInput({ initialValue = "", children }) {
 let [value, setValue] = useState(initialValue)
```

```
 function clear() {
 setValue("")
 }

 function bind() {
 return {
 value,
 onChange: (e) => setValue(e.target.value)
 }
 }

 return children({ value, setValue, clear, bind })
}

function SearchInput() {
 return (
 <ControlledInput initialValue="">
 {({ value, clear, bind }) => (
 <div>
 <input {...bind()} placeholder="Search..." />
 {value && (
 <button onClick={clear}>Clear</button>
 )}
 {value && <p>Searching for: {value}</p>}
 </div>
 )}
 </ControlledInput>
 )
}
```

COMBINING RENDER PROPS

```
function App() {
 return (
 <Toggle>
 {({ on, toggle }) => (
 <Fetch url="/api/data">
```

```
{({ data, loading }) => (
<div>
<button onClick={toggle}>
{on ? "Hide" : "Show"} Data
</button>

{on && (
loading
? <p>Loading...</p>
: <pre>{JSON.stringify(data, null, 2)}</pre>
)}
</div>
)}
</Fetch>
)}
</Toggle>
)
}
```

RENDER PROPS VS HOOKS

Render props can be replaced by hooks:

```
function useMousePosition() {
let [position, setPosition] = useState({ x: 0, y: 0 })

useEffect(() => {
function handleMouseMove(event) {
setPosition({ x: event.clientX, y: event.clientY })
}

window.addEventListener("mousemove", handleMouseMove)
return () => window.removeEventListener("mousemove",
handleMouseMove)
}, [])

return position
```

```
}

function App() {
 let { x, y } = useMousePosition()

 return <p>Mouse: {x}, {y}</p>
}
```

Hooks are often simpler than render props.
But render props still useful for:
- Component composition
- Callback-based APIs
- Library flexibility

PRACTICE

Build a mouse tracker render prop.

Create a data fetcher render prop.

Build a toggle render prop.

Create a form state render prop.

Compare with hook alternatives.

WHAT YOU LEARNED

You learned render props pattern.
You learned children as function.
You learned common use cases.
You learned combining render props.
You learned render props vs hooks.

The next chapter teaches higher-order components.
BOOK 17: REACT ADVANCED
Chapter 5: Higher-Order Components

WHAT ARE HOCS

A Higher-Order Component (HOC) is a function that:
- Takes a component
- Returns a new component with extra functionality

```
function withExtraProps(WrappedComponent) {
 return function EnhancedComponent(props) {
 return <WrappedComponent {...props} extraProp="value" />
 }
}
```

HOCs are a pattern for reusing logic.

BASIC HOC

```
function withLogging(WrappedComponent) {
 return function WithLogging(props) {
 useEffect(() => {
 console.log(`${WrappedComponent.name} mounted`)
 return () => console.log(`${WrappedComponent.name}
unmounted`)
 }, [])

 return <WrappedComponent {...props} />
 }
}

function MyComponent({ name }) {
```

```
 return <div>Hello, {name}!</div>
}

let MyComponentWithLogging = withLogging(MyComponent)
```

WITH AUTHENTICATION HOC

```
function withAuth(WrappedComponent) {
 return function WithAuth(props) {
 let { user, loading } = useAuth()

 if (loading) {
 return <div>Loading...</div>
 }

 if (!user) {
 return <Navigate to="/login" />
 }

 return <WrappedComponent {...props} user={user} />
 }
}

function Dashboard({ user }) {
 return <h1>Welcome, {user.name}!</h1>
}

let ProtectedDashboard = withAuth(Dashboard)
```

WITH LOADING HOC

```
function withLoading(WrappedComponent, loadingMessage =
"Loading...") {
 return function WithLoading({ isLoading, ...props }) {
```

```
 if (isLoading) {
 return <div className="loading">{loadingMessage}</div>
 }

 return <WrappedComponent {...props} />
 }
}

function UserList({ users }) {
 return (
 <ul>
 {users.map(user => (
 <li key={user.id}>{user.name}</li>
 ))}
 </ul>
 )
}

let UserListWithLoading = withLoading(UserList)

function App() {
 let [users, setUsers] = useState([])
 let [loading, setLoading] = useState(true)

 useEffect(() => {
 fetch("/api/users")
 .then(res => res.json())
 .then(data => {
 setUsers(data)
 setLoading(false)
 })
 }, [])

 return <UserListWithLoading isLoading={loading} users={users}
/>
}
```

WITH ERROR BOUNDARY HOC

```
function withErrorBoundary(WrappedComponent,
FallbackComponent) {
 return class WithErrorBoundary extends Component {
 constructor(props) {
 super(props)
 this.state = { hasError: false, error: null }
 }

 static getDerivedStateFromError(error) {
 return { hasError: true, error }
 }

 componentDidCatch(error, errorInfo) {
 console.error("Error caught:", error, errorInfo)
 }

 render() {
 if (this.state.hasError) {
 return FallbackComponent
 ? <FallbackComponent error={this.state.error} />
 : <div>Something went wrong</div>
 }

 return <WrappedComponent {...this.props} />
 }
 }
}

function ErrorFallback({ error }) {
 return <div>Error: {error.message}</div>
}

let SafeComponent = withErrorBoundary(RiskyComponent,
ErrorFallback)
```

WITH DATA FETCHING HOC

```
function withData(WrappedComponent, fetchFn) {
 return function WithData(props) {
 let [data, setData] = useState(null)
 let [loading, setLoading] = useState(true)
 let [error, setError] = useState(null)

 useEffect(() => {
 let cancelled = false

 async function loadData() {
 try {
 let result = await fetchFn(props)
 if (!cancelled) {
 setData(result)
 }
 } catch (err) {
 if (!cancelled) {
 setError(err.message)
 }
 } finally {
 if (!cancelled) {
 setLoading(false)
 }
 }
 }

 loadData()

 return () => {
 cancelled = true
 }
 }, [])

 if (loading) return <div>Loading...</div>
 if (error) return <div>Error: {error}</div>

 return <WrappedComponent {...props} data={data} />
 }
```

```
}

function UserProfile({ data: user }) {
 return (
 <div>
 <h1>{user.name}</h1>
 <p>{user.email}</p>
 </div>
 )
}

let UserProfileWithData = withData(
 UserProfile,
 (props) => fetch(`/api/users/${props.userId}`).then(r =>
r.json())
)
```

WITH THEME HOC

```
function withTheme(WrappedComponent) {
 return function WithTheme(props) {
 let theme = useTheme()

 return <WrappedComponent {...props} theme={theme} />
 }
}

function Button({ theme, children, ...props }) {
 return (
 <button
 style={{
 backgroundColor: theme.primaryColor,
 color: theme.textColor
 }}
 {...props}
 >
 {children}
```

```
 </button>
 )
}

let ThemedButton = withTheme(Button)
```

COMPOSING HOCS

Combine multiple HOCs:

```
function compose(...fns) {
 return (x) => fns.reduceRight((acc, fn) => fn(acc), x)
}

let EnhancedComponent = compose(
 withAuth,
 withTheme,
 withLogging
)(MyComponent)
```

Or manually:

```
let EnhancedComponent =
withAuth(withTheme(withLogging(MyComponent)))
```

HOC CONVENTIONS

1. Pass through props:

```
function withEnhancement(WrappedComponent) {
 return function Enhanced(props) {
 return <WrappedComponent {...props} enhancement="value" />
 }
```

```
}
```

2. Preserve display name:

```
function withEnhancement(WrappedComponent) {
 function Enhanced(props) {
 return <WrappedComponent {...props} />
 }

 Enhanced.displayName = `withEnhancement(${
 WrappedComponent.displayName || WrappedComponent.name ||
"Component"
 })`

 return Enhanced
}
```

3. Copy static methods:

```
import hoistNonReactStatics from "hoist-non-react-statics"

function withEnhancement(WrappedComponent) {
 function Enhanced(props) {
 return <WrappedComponent {...props} />
 }

 hoistNonReactStatics(Enhanced, WrappedComponent)
 return Enhanced
}
```

4. Forward refs:

```
import { forwardRef } from "react"

function withEnhancement(WrappedComponent) {
 let Enhanced = forwardRef((props, ref) => {
 return <WrappedComponent ref={ref} {...props} />
```

```
 })

 Enhanced.displayName =
`withEnhancement(${WrappedComponent.displayName ||
WrappedComponent.name})`
 return Enhanced
}
```

HOCS VS HOOKS

HOCs were popular before hooks.
Many HOC patterns can be replaced by hooks:

HOC:

```
let ThemedButton = withTheme(Button)
```

Hook:

```
function Button() {
 let theme = useTheme()
 return <button style={{ color: theme.color }}>...</button>
}
```

Hooks are often simpler.
HOCs still useful for:
- Wrapping third-party components
- Adding behavior without modifying components
- Cross-cutting concerns

PRACTICE

Build withAuth HOC.

Create withLoading HOC.

Build withData HOC.

Compose multiple HOCs.

Compare with hook alternatives.

WHAT YOU LEARNED

You learned HOC pattern.
You learned common HOC use cases.
You learned HOC conventions.
You learned to compose HOCs.
You learned HOCs vs hooks.

The next chapter teaches advanced custom hooks.
BOOK 17: REACT ADVANCED
Chapter 6: Custom Hooks Advanced

COMPLEX STATE HOOKS

useAsync hook for async operations:

```
function useAsync(asyncFunction, immediate = true) {
 let [state, setState] = useState({
 status: "idle",
 data: null,
 error: null
 })
```

```
 let execute = useCallback(async (...args) => {
 setState({ status: "pending", data: null, error: null })

 try {
 let data = await asyncFunction(...args)
 setState({ status: "success", data, error: null })
 return data
 } catch (error) {
 setState({ status: "error", data: null, error })
 throw error
 }
 }, [asyncFunction])

 useEffect(() => {
 if (immediate) {
 execute()
 }
 }, [execute, immediate])

 return { ...state, execute }
}

function UserProfile({ userId }) {
 let { status, data: user, error } = useAsync(
 () => fetch(`/api/users/${userId}`).then(r => r.json())
 )

 if (status === "pending") return <p>Loading...</p>
 if (status === "error") return <p>Error: {error.message}</p>
 if (!user) return null

 return <h1>{user.name}</h1>
}
```

USEMEDIA HOOK

Responsive design hook:

```
function useMedia(query) {
 let [matches, setMatches] = useState(() => 
 window.matchMedia(query).matches
 )

 useEffect(() => {
 let mediaQuery = window.matchMedia(query)

 function handleChange(event) {
 setMatches(event.matches)
 }

 mediaQuery.addEventListener("change", handleChange)
 return () => mediaQuery.removeEventListener("change",
handleChange)
 }, [query])

 return matches
}

function useBreakpoint() {
 let isMobile = useMedia("(max-width: 767px)")
 let isTablet = useMedia("(min-width: 768px) and (max-width:
1023px)")
 let isDesktop = useMedia("(min-width: 1024px)")

 return { isMobile, isTablet, isDesktop }
}

function ResponsiveComponent() {
 let { isMobile, isDesktop } = useBreakpoint()

 return (
 <div>
 {isMobile && <MobileNav />}
 {isDesktop && <DesktopNav />}
 </div>
 )
}
```

USEINTERSECTIONOBSERVER HOOK

Detect element visibility:

```
function useIntersectionObserver(ref, options = {}) {
 let [isIntersecting, setIsIntersecting] = useState(false)

 useEffect(() => {
 let element = ref.current
 if (!element) return

 let observer = new IntersectionObserver(
 ([entry]) => {
 setIsIntersecting(entry.isIntersecting)
 },
 {
 threshold: options.threshold || 0,
 root: options.root || null,
 rootMargin: options.rootMargin || "0px"
 }
 )

 observer.observe(element)

 return () => observer.disconnect()
 }, [ref, options.threshold, options.root, options.rootMargin])

 return isIntersecting
}

function LazyImage({ src, alt }) {
 let ref = useRef(null)
 let isVisible = useIntersectionObserver(ref, { threshold: 0.1 })
 let [loaded, setLoaded] = useState(false)

 return (
```

```
 <div ref={ref}>
 {(isVisible || loaded) && (
 <img
 src={src}
 alt={alt}
 onLoad={() => setLoaded(true)}
 />
 )}
 </div>
 )
}
```

USEUNDOREDO HOOK

```
function useUndoRedo(initialState) {
 let [states, setStates] = useState([initialState])
 let [index, setIndex] = useState(0)

 let state = states[index]

 let set = useCallback((newState) => {
 let newStates = states.slice(0, index + 1)
 newStates.push(typeof newState === "function" ?
newState(state) : newState)
 setStates(newStates)
 setIndex(newStates.length - 1)
 }, [states, index, state])

 let undo = useCallback(() => {
 if (index > 0) {
 setIndex(index - 1)
 }
 }, [index])

 let redo = useCallback(() => {
 if (index < states.length - 1) {
 setIndex(index + 1)
```

```
 }
 }, [index, states.length])

 let canUndo = index > 0
 let canRedo = index < states.length - 1

 return { state, set, undo, redo, canUndo, canRedo }
}

function TextEditor() {
 let { state: text, set: setText, undo, redo, canUndo, canRedo } =
 useUndoRedo("")

 return (
 <div>
 <textarea
 value={text}
 onChange={(e) => setText(e.target.value)}
 />
 <button onClick={undo} disabled={!canUndo}>Undo</
button>
 <button onClick={redo} disabled={!canRedo}>Redo</
button>
 </div>
 )
}
```

USEWEBSOCKET HOOK

```
function useWebSocket(url) {
 let [socket, setSocket] = useState(null)
 let [messages, setMessages] = useState([])
 let [status, setStatus] = useState("disconnected")

 useEffect(() => {
 let ws = new WebSocket(url)
```

```
 ws.onopen = () => {
 setStatus("connected")
 setSocket(ws)
 }

 ws.onmessage = (event) => {
 setMessages(prev => [...prev, JSON.parse(event.data)])
 }

 ws.onclose = () => {
 setStatus("disconnected")
 setSocket(null)
 }

 ws.onerror = () => {
 setStatus("error")
 }

 return () => ws.close()
 }, [url])

 let send = useCallback((data) => {
 if (socket && socket.readyState === WebSocket.OPEN) {
 socket.send(JSON.stringify(data))
 }
 }, [socket])

 return { messages, send, status }
}

function Chat() {
 let { messages, send, status } = useWebSocket("wss://
chat.example.com")
 let [input, setInput] = useState("")

 function handleSend() {
 send({ type: "message", text: input })
 setInput("")
 }
```

```
 return (
 <div>
 <p>Status: {status}</p>
 <ul>
 {messages.map((msg, i) => (
 <li key={i}>{msg.text}</li>
 ))}
 </ul>
 <input value={input} onChange={(e) =>
setInput(e.target.value)} />
 <button onClick={handleSend}>Send</button>
 </div>
 )
}
```

USEINTERVAL HOOK

```
function useInterval(callback, delay) {
 let savedCallback = useRef()

 useEffect(() => {
 savedCallback.current = callback
 }, [callback])

 useEffect(() => {
 if (delay === null) return

 let id = setInterval(() => {
 savedCallback.current()
 }, delay)

 return () => clearInterval(id)
 }, [delay])
}

function Timer() {
 let [count, setCount] = useState(0)
```

```
 let [isRunning, setIsRunning] = useState(false)

 useInterval(
 () => setCount(c => c + 1),
 isRunning ? 1000 : null
 )

 return (
 <div>
 <p>Count: {count}</p>
 <button onClick={() => setIsRunning(!isRunning)}>
 {isRunning ? "Pause" : "Start"}
 </button>
 </div>
 )
}
```

USECOPYTOCLIPBOARD HOOK

```
function useCopyToClipboard() {
 let [copied, setCopied] = useState(false)

 let copy = useCallback(async (text) => {
 try {
 await navigator.clipboard.writeText(text)
 setCopied(true)
 setTimeout(() => setCopied(false), 2000)
 return true
 } catch (err) {
 setCopied(false)
 return false
 }
 }, [])

 return { copy, copied }
}
```

```
function ShareButton({ text }) {
 let { copy, copied } = useCopyToClipboard()

 return (
 <button onClick={() => copy(text)}>
 {copied ? "Copied!" : "Copy to clipboard"}
 </button>
 )
}
```

USEEVENTLISTENER HOOK

```
function useEventListener(eventName, handler, element =
window) {
 let savedHandler = useRef()

 useEffect(() => {
 savedHandler.current = handler
 }, [handler])

 useEffect(() => {
 let targetElement = element?.current || element
 if (!targetElement?.addEventListener) return

 let eventListener = (event) => savedHandler.current(event)

 targetElement.addEventListener(eventName, eventListener)
 return () => targetElement.removeEventListener(eventName,
eventListener)
 }, [eventName, element])
}

function ScrollTracker() {
 let [scrollY, setScrollY] = useState(0)

 useEventListener("scroll", () => {
 setScrollY(window.scrollY)
```

```
 })

 return <p>Scroll position: {scrollY}</p>
}
```

USEKEYPRESS HOOK

```
function useKeyPress(targetKey) {
 let [keyPressed, setKeyPressed] = useState(false)

 useEffect(() => {
 function downHandler({ key }) {
 if (key === targetKey) {
 setKeyPressed(true)
 }
 }

 function upHandler({ key }) {
 if (key === targetKey) {
 setKeyPressed(false)
 }
 }

 window.addEventListener("keydown", downHandler)
 window.addEventListener("keyup", upHandler)

 return () => {
 window.removeEventListener("keydown", downHandler)
 window.removeEventListener("keyup", upHandler)
 }
 }, [targetKey])

 return keyPressed
}

function Game() {
 let spacePressed = useKeyPress(" ")
```

```
 let escPressed = useKeyPress("Escape")

 return (
  <div>
  {spacePressed && <p>Jumping!</p>}
  {escPressed && <p>Paused</p>}
  </div>
 )
}
```

PRACTICE

Build useAsync hook.

Create useUndoRedo hook.

Build useWebSocket hook.

Create useEventListener hook.

Combine hooks for complex features.

WHAT YOU LEARNED

You learned complex custom hooks.
You learned useAsync pattern.
You learned observer hooks.
You learned undo/redo pattern.
You learned WebSocket hook.
You learned event listener hooks.

The next chapter teaches portals.
BOOK 17: REACT ADVANCED
Chapter 7: Portals

WHAT ARE PORTALS

Portals render children outside parent DOM hierarchy.

Useful for:
- Modals
- Tooltips
- Dropdowns
- Notifications
- Full-screen overlays

Child appears in different DOM location but behaves like normal React child.

CREATING A PORTAL

```
import { createPortal } from "react-dom"

function Modal({ children }) {
 return createPortal(
 <div className="modal-overlay">
 <div className="modal-content">
 {children}
 </div>
 </div>,
 document.body
 )
}

function App() {
 let [showModal, setShowModal] = useState(false)
```

```
 return (
 <div className="app">
 <button onClick={() => setShowModal(true)}>
 Open Modal
 </button>

 {showModal && (
 <Modal>
 <h2>Modal Title</h2>
 <p>Modal content here</p>
 <button onClick={() => setShowModal(false)}>
 Close
 </button>
 </Modal>
 )}
 </div>
 )
}
```

Modal renders in document.body, not inside .app.

PORTAL CONTAINER

Create dedicated portal root:

index.html:

```
 <body>
 <div id="root"></div>
 <div id="portal-root"></div>
 </body>
```

Use portal root:

```
function Portal({ children }) {
 let portalRoot = document.getElementById("portal-root")
```

```
 return createPortal(children, portalRoot)
}
```

MODAL COMPONENT

```
function Modal({ isOpen, onClose, children }) {
 useEffect(() => {
 if (isOpen) {
 document.body.style.overflow = "hidden"
 }

 return () => {
 document.body.style.overflow = ""
 }
 }, [isOpen])

 useEffect(() => {
 function handleEscape(event) {
 if (event.key === "Escape") {
 onClose()
 }
 }

 if (isOpen) {
 document.addEventListener("keydown", handleEscape)
 }

 return () => {
 document.removeEventListener("keydown", handleEscape)
 }
 }, [isOpen, onClose])

 if (!isOpen) return null

 return createPortal(
 <div
 className="modal-overlay"
```

```
 onClick={(e) => {
 if (e.target === e.currentTarget) {
 onClose()
 }
 }}
 >
 <div className="modal-content" role="dialog" aria-
modal="true">
 <button className="modal-close" onClick={onClose}>
 &times;
 </button>
 {children}
 </div>
 </div>,
 document.body
 )
}

function App() {
 let [isOpen, setIsOpen] = useState(false)

 return (
 <div>
 <button onClick={() => setIsOpen(true)}>Open Modal</
button>

 <Modal isOpen={isOpen} onClose={() => setIsOpen(false)}>
 <h2>Modal Title</h2>
 <p>This is modal content.</p>
 </Modal>
 </div>
 )
}
```

TOOLTIP COMPONENT

```
function Tooltip({ children, content, position = "top" }) {
```

```
let [isVisible, setIsVisible] = useState(false)
let [coords, setCoords] = useState({ top: 0, left: 0 })
let triggerRef = useRef(null)

function updatePosition() {
if (!triggerRef.current) return

let rect = triggerRef.current.getBoundingClientRect()

let positions = {
top: {
top: rect.top - 10,
left: rect.left + rect.width / 2
},
bottom: {
top: rect.bottom + 10,
left: rect.left + rect.width / 2
},
left: {
top: rect.top + rect.height / 2,
left: rect.left - 10
},
right: {
top: rect.top + rect.height / 2,
left: rect.right + 10
}
}

setCoords(positions[position])
}

function handleMouseEnter() {
updatePosition()
setIsVisible(true)
}

function handleMouseLeave() {
setIsVisible(false)
}
```

```
 return (
 < >
 <span
 ref={triggerRef}
 onMouseEnter={handleMouseEnter}
 onMouseLeave={handleMouseLeave}
 >
 {children}
 </span>

 {isVisible && createPortal(
 <div
 className={`tooltip tooltip-${position}`}
 style={{
 position: "fixed",
 top: coords.top,
 left: coords.left,
 transform: position === "top" || position === "bottom"
 ? "translateX(-50%)"
 : "translateY(-50%)"
 }}
 >
 {content}
 </div>,
 document.body
 )}
 </>
 )
}

function App() {
 return (
 <p>
 Hover over this{" "}
 <Tooltip content="This is a tooltip!">
 <span className="highlighted">word</span>
 </Tooltip>{" "}
 to see a tooltip.
 </p>
 )
```

```
}
```

DROPDOWN MENU

```
function Dropdown({ trigger, children }) {
 let [isOpen, setIsOpen] = useState(false)
 let [position, setPosition] = useState({ top: 0, left: 0 })
 let triggerRef = useRef(null)
 let dropdownRef = useRef(null)

 useEffect(() => {
 function handleClickOutside(event) {
 if (
 dropdownRef.current &&
 !dropdownRef.current.contains(event.target) &&
 !triggerRef.current.contains(event.target)
 ) {
 setIsOpen(false)
 }
 }

 document.addEventListener("mousedown", handleClickOutside)
 return () => document.removeEventListener("mousedown",
handleClickOutside)
 }, [])

 function handleTriggerClick() {
 if (!triggerRef.current) return

 let rect = triggerRef.current.getBoundingClientRect()
 setPosition({
 top: rect.bottom + 5,
 left: rect.left
 })
 setIsOpen(!isOpen)
 }
```

```
 return (
 <>
 <div ref={triggerRef} onClick={handleTriggerClick}>
 {trigger}
 </div>

 {isOpen && createPortal(
 <div
 ref={dropdownRef}
 className="dropdown-menu"
 style={{
 position: "fixed",
 top: position.top,
 left: position.left
 }}
 >
 {children}
 </div>,
 document.body
 )}
 </>
 )
}

function App() {
 return (
 <Dropdown trigger={<button>Options</button>}>
 <ul>
 <li onClick={() => console.log("Edit")}>Edit</li>
 <li onClick={() => console.log("Delete")}>Delete</li>
 <li onClick={() => console.log("Share")}>Share</li>
 </ul>
 </Dropdown>
 )
}
```

TOAST NOTIFICATIONS

```
let ToastContext = createContext(null)

function ToastProvider({ children }) {
 let [toasts, setToasts] = useState([])

 function addToast(message, type = "info", duration = 3000) {
 let id = Date.now()
 setToasts(prev => [...prev, { id, message, type }])

 setTimeout(() => {
 removeToast(id)
 }, duration)
 }

 function removeToast(id) {
 setToasts(prev => prev.filter(t => t.id !== id))
 }

 return (
 <ToastContext.Provider value={{ addToast }}>
 {children}

 {createPortal(
 <div className="toast-container">
 {toasts.map(toast => (
 <div
 key={toast.id}
 className={`toast toast-${toast.type}`}
 >
 {toast.message}
 <button onClick={() => removeToast(toast.id)}>
 &times;
 </button>
 </div>
 ))}
 </div>,
 document.body
 )}
 </ToastContext.Provider>
```

```
 )
}

function useToast() {
 return useContext(ToastContext)
}

function App() {
 let { addToast } = useToast()

 return (
  <button onClick={() => addToast("Operation successful!",
"success")}>
  Show Toast
  </button>
 )
}
```

EVENT BUBBLING

Portal children bubble events to React parent:

```
function Parent() {
 function handleClick() {
 console.log("Clicked inside portal")
 }

 return (
 <div onClick={handleClick}>
 <Modal>
 <button>Click me</button>
 </Modal>
 </div>
 )
}
```

Even though Modal renders outside Parent in DOM, click bubbles to

Parent in React tree.

PRACTICE

Build a Modal component.

Create Tooltip with positioning.

Build Dropdown menu.

Create Toast notification system.

Handle click outside and escape key.

WHAT YOU LEARNED

You learned what portals are.
You learned createPortal syntax.
You learned Modal component.
You learned Tooltip positioning.
You learned Dropdown menus.
You learned Toast notifications.

The next chapter teaches Suspense and concurrent features.
BOOK 17: REACT ADVANCED
Chapter 8: Suspense and Concurrent Features

WHAT IS SUSPENSE

Suspense lets components wait for something before rendering.

Shows fallback while waiting:
- Code loading (lazy)
- Data fetching (with libraries)

```
import { Suspense, lazy } from "react"

let Dashboard = lazy(() => import("./Dashboard"))

function App() {
 return (
 <Suspense fallback={<p>Loading...</p>}>
 <Dashboard />
 </Suspense>
 )
}
```

LAZY LOADING COMPONENTS

Split code into chunks:

```
import { lazy, Suspense } from "react"

let Home = lazy(() => import("./pages/Home"))
let About = lazy(() => import("./pages/About"))
let Dashboard = lazy(() => import("./pages/Dashboard"))

function App() {
 return (
 <BrowserRouter>
 <Suspense fallback={<LoadingSpinner />}>
 <Routes>
 <Route path="/" element={<Home />} />
 <Route path="/about" element={<About />} />
 <Route path="/dashboard" element={<Dashboard />} />
 </Routes>
 </Suspense>
```

```
 </BrowserRouter>
 )
}
```

Each route loads only when visited.

MULTIPLE SUSPENSE BOUNDARIES

Granular loading states:

```
function App() {
 return (
 <div>
 <Suspense fallback={<p>Loading header...</p>}>
 <Header />
 </Suspense>

 <main>
 <Suspense fallback={<p>Loading sidebar...</p>}>
 <Sidebar />
 </Suspense>

 <Suspense fallback={<p>Loading content...</p>}>
 <Content />
 </Suspense>
 </main>
 </div>
 )
}
```

Each section shows its own loading state.

NESTED SUSPENSE

```
function App() {
 return (
 <Suspense fallback={<p>Loading app...</p>}>
 <Dashboard />
 </Suspense>
 )
}

function Dashboard() {
 return (
 <div>
 <h1>Dashboard</h1>
 <Suspense fallback={<p>Loading charts...</p>}>
 <Charts />
 </Suspense>
 <Suspense fallback={<p>Loading table...</p>}>
 <DataTable />
 </Suspense>
 </div>
 )
}
```

Parent suspense shows while Dashboard loads.
Child suspenses show while charts/table load.

DATA FETCHING WITH SUSPENSE

Using React Query with Suspense:

```
import { QueryClient, QueryClientProvider } from "@tanstack/
react-query"

let queryClient = new QueryClient({
 defaultOptions: {
 queries: {
 suspense: true
 }
```

```
 }
})

function App() {
 return (
 <QueryClientProvider client={queryClient}>
 <Suspense fallback={<p>Loading...</p>}>
 <UserList />
 </Suspense>
 </QueryClientProvider>
 )
}

function UserList() {
 let { data: users } = useQuery({
 queryKey: ["users"],
 queryFn: () => fetch("/api/users").then(r => r.json())
 })

 return (
 <ul>
 {users.map(user => (
 <li key={user.id}>{user.name}</li>
 ))}
 </ul>
 )
}
```

USETRANSITION

Mark state updates as non-urgent:

```
import { useState, useTransition } from "react"

function SearchResults() {
 let [query, setQuery] = useState("")
 let [isPending, startTransition] = useTransition()
```

```
 let [results, setResults] = useState([])

 function handleChange(e) {
 let value = e.target.value
 setQuery(value)

 startTransition(() => {
 let filtered = searchData(value)
 setResults(filtered)
 })
 }

 return (
 <div>
 <input value={query} onChange={handleChange} />

 {isPending && <p>Searching...</p>}

 <ul>
 {results.map(item => (
 <li key={item.id}>{item.name}</li>
 ))}
 </ul>
 </div>
 )
}
```

Input stays responsive while results update.

USEDEFERREDVALUE

Defer a value to avoid blocking:

```
import { useState, useDeferredValue, useMemo } from "react"

function SearchResults() {
 let [query, setQuery] = useState("")
```

```
 let deferredQuery = useDeferredValue(query)

 let results = useMemo(() => {
 return searchData(deferredQuery)
 }, [deferredQuery])

 let isStale = query !== deferredQuery

 return (
 <div>
 <input
 value={query}
 onChange={(e) => setQuery(e.target.value)}
 />

 <ul style={{ opacity: isStale ? 0.5 : 1 }}>
 {results.map(item => (
 <li key={item.id}>{item.name}</li>
 ))}
 </ul>
 </div>
 )
}
```

Results update with delay, keeping input responsive.

ERROR HANDLING WITH SUSPENSE

Combine Suspense with Error Boundaries:

```
import { ErrorBoundary } from "react-error-boundary"

function App() {
 return (
 <ErrorBoundary
 fallback={<p>Something went wrong</p>}
 onReset={() => window.location.reload()}
```

```
 >
 <Suspense fallback={<p>Loading...</p>}>
 <Dashboard />
 </Suspense>
 </ErrorBoundary>
 )
}
```

STARTVIEWTRANSITION

Smooth visual transitions:

```
import { startTransition } from "react"

function TabPanel() {
 let [tab, setTab] = useState("home")

 function handleTabClick(newTab) {
 startTransition(() => {
 setTab(newTab)
 })
 }

 return (
 <div>
 <nav>
 <button onClick={() => handleTabClick("home")}>Home</
button>
 <button onClick={() => handleTabClick("about")}>About</
button>
 </nav>

 <Suspense fallback={<p>Loading tab...</p>}>
 {tab === "home" && <HomeTab />}
 {tab === "about" && <AboutTab />}
 </Suspense>
 </div>
```

```
 )
}
```

SUSPENSE LIST

Control reveal order of multiple suspense boundaries:

```
import { SuspenseList, Suspense } from "react"

function UserProfile() {
 return (
  <SuspenseList revealOrder="forwards">
  <Suspense fallback={<p>Loading header...</p>}>
  <ProfileHeader />
  </Suspense>
  <Suspense fallback={<p>Loading bio...</p>}>
  <ProfileBio />
  </Suspense>
  <Suspense fallback={<p>Loading posts...</p>}>
  <ProfilePosts />
  </Suspense>
  </SuspenseList>
 )
}
```

revealOrder options:
- forwards: Top to bottom
- backwards: Bottom to top
- together: All at once

STREAMING WITH SUSPENSE

Server-side rendering with streaming:

```
function App() {
 return (
 <html>
 <head>
 <title>My App</title>
 </head>
 <body>
 <Suspense fallback={<p>Loading nav...</p>}>
 <Nav />
 </Suspense>

 <Suspense fallback={<p>Loading content...</p>}>
 <Content />
 </Suspense>
 </body>
 </html>
 )
}
```

Server sends HTML progressively as data becomes available.

BEST PRACTICES

1. Place Suspense at right level:
 Too high: everything shows loading
 Too low: too many loading states

2. Provide meaningful fallbacks:
 Show skeleton loaders
 Match layout of loaded content

3. Use transitions for user actions:
 Keep UI responsive
 Show pending state

4. Handle errors:
 Wrap Suspense in ErrorBoundary

Provide recovery options

5. Test loading states:
Use React DevTools to suspend
Verify all loading UI

PRACTICE

Implement lazy loading for routes.

Use multiple Suspense boundaries.

Apply useTransition for search.

Use useDeferredValue for lists.

Combine with error boundaries.

WHAT YOU LEARNED

You learned Suspense for code splitting.
You learned Suspense boundaries.
You learned useTransition.
You learned useDeferredValue.
You learned error handling with Suspense.
You learned best practices.

The next chapter teaches testing React components.
BOOK 17: REACT ADVANCED
Chapter 9: Testing React Components

WHY TEST REACT COMPONENTS

Tests verify:
- Components render correctly
- User interactions work
- State updates properly
- Edge cases handled

Tests prevent regressions.

TESTING TOOLS

Jest: Test runner and assertions
React Testing Library: Component testing
Vitest: Fast test runner (Vite projects)

npm install -D jest @testing-library/react @testing-library/jest-dom

Or with Vitest:

npm install -D vitest @testing-library/react @testing-library/jest-dom

BASIC COMPONENT TEST

Button.jsx:

```
function Button({ onClick, children }) {
 return (
 <button onClick={onClick} className="btn">
 {children}
 </button>
```

```
 )
}

export default Button
```

Button.test.jsx:

```
import { render, screen, fireEvent } from "@testing-library/react"
import Button from "./Button"

test("renders button with text", () => {
 render(<Button>Click me</Button>)

 let button = screen.getByRole("button", { name: /click me/i })
 expect(button).toBeInTheDocument()
})

test("calls onClick when clicked", () => {
 let handleClick = jest.fn()
 render(<Button onClick={handleClick}>Click</Button>)

 let button = screen.getByRole("button")
 fireEvent.click(button)

 expect(handleClick).toHaveBeenCalledTimes(1)
})
```

QUERIES

Finding elements:

getByRole - Preferred for accessibility
getByText - Find by text content
getByLabelText - Form inputs by label
getByPlaceholderText - By placeholder
getByTestId - By data-testid attribute

Variants:

getBy - Throws if not found
queryBy - Returns null if not found
findBy - Returns promise (for async)

```
screen.getByRole("button")
screen.getByText("Hello")
screen.getByLabelText("Email")
screen.queryByRole("dialog")
await screen.findByText("Loaded")
```

TESTING USER EVENTS

```
import userEvent from "@testing-library/user-event"

test("user can type in input", async () => {
 let user = userEvent.setup()
 render(<Form />)

 let input = screen.getByLabelText("Name")
 await user.type(input, "Alice")

 expect(input).toHaveValue("Alice")
})

test("user can click button", async () => {
 let user = userEvent.setup()
 let handleClick = jest.fn()
 render(<Button onClick={handleClick}>Click</Button>)

 await user.click(screen.getByRole("button"))

 expect(handleClick).toHaveBeenCalled()
```

```
})

test("user can select option", async () => {
 let user = userEvent.setup()
 render(<Select />)

 await user.selectOptions(screen.getByRole("combobox"), "option2")

 expect(screen.getByRole("combobox")).toHaveValue("option2")
})
```

TESTING STATE CHANGES

```
function Counter() {
 let [count, setCount] = useState(0)

 return (
 <div>
 <p>Count: {count}</p>
 <button onClick={() => setCount(count + 1)}>Increment</
button>
 </div>
 )
}

test("increments counter on click", async () => {
 let user = userEvent.setup()
 render(<Counter />)

 expect(screen.getByText("Count: 0")).toBeInTheDocument()

 await user.click(screen.getByRole("button", { name: /increment/i
}))

 expect(screen.getByText("Count: 1")).toBeInTheDocument()
})
```

TESTING FORMS

```
function LoginForm({ onSubmit }) {
 let [email, setEmail] = useState("")
 let [password, setPassword] = useState("")

 function handleSubmit(e) {
 e.preventDefault()
 onSubmit({ email, password })
 }

 return (
 <form onSubmit={handleSubmit}>
 <label>
 Email
 <input
 type="email"
 value={email}
 onChange={(e) => setEmail(e.target.value)}
 />
 </label>
 <label>
 Password
 <input
 type="password"
 value={password}
 onChange={(e) => setPassword(e.target.value)}
 />
 </label>
 <button type="submit">Login</button>
 </form>
 )
}

test("submits form with values", async () => {
 let user = userEvent.setup()
 let handleSubmit = jest.fn()
```

```
 render(<LoginForm onSubmit={handleSubmit} />)

 await user.type(screen.getByLabelText(/email/i),
"test@example.com")
 await user.type(screen.getByLabelText(/password/i),
"password123")
 await user.click(screen.getByRole("button", { name: /login/i }))

 expect(handleSubmit).toHaveBeenCalledWith({
 email: "test@example.com",
 password: "password123"
 })
})
```

TESTING ASYNC BEHAVIOR

```
function UserList() {
 let [users, setUsers] = useState([])
 let [loading, setLoading] = useState(true)

 useEffect(() => {
 fetch("/api/users")
 .then(res => res.json())
 .then(data => {
 setUsers(data)
 setLoading(false)
 })
 }, [])

 if (loading) return <p>Loading...</p>

 return (
 <ul>
 {users.map(user => (
 <li key={user.id}>{user.name}</li>
 ))}
 </ul>
```

```
 )
}

test("loads and displays users", async () => {
 global.fetch = jest.fn(() =>
 Promise.resolve({
 json: () => Promise.resolve([
 { id: 1, name: "Alice" },
 { id: 2, name: "Bob" }
 ])
 })
 )

 render(<UserList />)

 expect(screen.getByText("Loading...")).toBeInTheDocument()

 await screen.findByText("Alice")
 expect(screen.getByText("Bob")).toBeInTheDocument()
})
```

MOCKING

Mock functions:

```
let mockFn = jest.fn()
mockFn.mockReturnValue("value")
mockFn.mockResolvedValue("async value")
mockFn.mockImplementation(() => "custom")
```

Mock modules:

```
jest.mock("./api", () => ({
 fetchUsers: jest.fn()
}))
```

Mock components:

```
jest.mock("./HeavyComponent", () => {
 return function MockHeavyComponent() {
 return <div data-testid="mock-heavy">Mocked</div>
 }
})
```

TESTING WITH CONTEXT

```
function TestWrapper({ children }) {
 return (
 <AuthProvider>
 <ThemeProvider>
 {children}
 </ThemeProvider>
 </AuthProvider>
 )
}

test("component with context", () => {
 render(<MyComponent />, { wrapper: TestWrapper })

 expect(screen.getByText("User Profile")).toBeInTheDocument()
})
```

Custom render:

```
import { render } from "@testing-library/react"

function customRender(ui, options) {
 return render(ui, { wrapper: TestWrapper, ...options })
}

export * from "@testing-library/react"
```

```
export { customRender as render }
```

TESTING HOOKS

```
import { renderHook, act } from "@testing-library/react"

function useCounter(initialValue = 0) {
 let [count, setCount] = useState(initialValue)

 function increment() {
 setCount(c => c + 1)
 }

 return { count, increment }
}

test("useCounter hook", () => {
 let { result } = renderHook(() => useCounter(5))

 expect(result.current.count).toBe(5)

 act(() => {
 result.current.increment()
 })

 expect(result.current.count).toBe(6)
})
```

SNAPSHOT TESTING

```
test("renders correctly", () => {
 let { container } = render(<Button>Click me</Button>)

 expect(container).toMatchSnapshot()
```

```
})
```

Creates snapshot file on first run.
Compares against snapshot on subsequent runs.
Update snapshots: jest --updateSnapshot

TEST STRUCTURE

```
describe("Button component", () => {
 test("renders with text", () => {
 render(<Button>Click</Button>)
 expect(screen.getByRole("button")).toBeInTheDocument()
 })

 test("applies className", () => {
 render(<Button className="primary">Click</Button>)
 expect(screen.getByRole("button")).toHaveClass("primary")
 })

 test("handles click", async () => {
 let user = userEvent.setup()
 let onClick = jest.fn()
 render(<Button onClick={onClick}>Click</Button>)

 await user.click(screen.getByRole("button"))

 expect(onClick).toHaveBeenCalled()
 })

 test("can be disabled", () => {
 render(<Button disabled>Click</Button>)
 expect(screen.getByRole("button")).toBeDisabled()
 })
})
```

PRACTICE

Write tests for Button component.

Test forms with user events.

Test async data loading.

Mock API calls.

Test custom hooks.

WHAT YOU LEARNED

You learned testing tools.
You learned queries and assertions.
You learned user event testing.
You learned async testing.
You learned mocking.
You learned hook testing.

The next chapter teaches state management libraries.
BOOK 17: REACT ADVANCED
Chapter 10: State Management Libraries

WHEN TO USE STATE LIBRARIES

Use React state when:
- Local component state
- Simple apps
- Few shared states

Use state libraries when:
- Complex global state
- Many components need same data
- Predictable state updates
- DevTools needed

ZUSTAND

Simple, flexible state management:

npm install zustand

```
import { create } from "zustand"

let useStore = create((set) => ({
 count: 0,
 increment: () => set((state) => ({ count: state.count + 1 })),
 decrement: () => set((state) => ({ count: state.count - 1 })),
 reset: () => set({ count: 0 })
}))

function Counter() {
 let { count, increment, decrement, reset } = useStore()

 return (
 <div>
 <p>Count: {count}</p>
 <button onClick={increment}>+</button>
 <button onClick={decrement}>-</button>
 <button onClick={reset}>Reset</button>
 </div>
 )
}
```

Selectors to avoid re-renders:

```
function CountDisplay() {
 let count = useStore((state) => state.count)
 return <p>Count: {count}</p>
}
```

Async actions:

```
let useStore = create((set) => ({
 users: [],
 loading: false,

 fetchUsers: async () => {
 set({ loading: true })
 let response = await fetch("/api/users")
 let users = await response.json()
 set({ users, loading: false })
 }
}))
```

Persist middleware:

```
import { persist } from "zustand/middleware"

let useStore = create(
 persist(
 (set) => ({
 theme: "light",
 setTheme: (theme) => set({ theme })
 }),
 { name: "theme-storage" }
 )
)
```

REDUX TOOLKIT

Official Redux approach:

npm install @reduxjs/toolkit react-redux

Create slice:

```
import { createSlice } from "@reduxjs/toolkit"

let counterSlice = createSlice({
 name: "counter",
 initialState: { value: 0 },
 reducers: {
 increment: (state) => {
 state.value += 1
 },
 decrement: (state) => {
 state.value -= 1
 },
 incrementByAmount: (state, action) => {
 state.value += action.payload
 }
 }
})

export let { increment, decrement, incrementByAmount } =
counterSlice.actions
export default counterSlice.reducer
```

Configure store:

```
import { configureStore } from "@reduxjs/toolkit"
import counterReducer from "./counterSlice"

let store = configureStore({
 reducer: {
 counter: counterReducer
 }
})
```

export default store

Provider:

```
import { Provider } from "react-redux"
import store from "./store"

function App() {
 return (
 <Provider store={store}>
 <Counter />
 </Provider>
 )
}
```

Use in component:

```
import { useSelector, useDispatch } from "react-redux"
import { increment, decrement } from "./counterSlice"

function Counter() {
 let count = useSelector((state) => state.counter.value)
 let dispatch = useDispatch()

 return (
 <div>
 <p>Count: {count}</p>
 <button onClick={() => dispatch(increment())}>+</button>
 <button onClick={() => dispatch(decrement())}>-</button>
 </div>
 )
}
```

Async thunks:

```
import { createAsyncThunk } from "@reduxjs/toolkit"
```

```
let fetchUsers = createAsyncThunk("users/fetch", async () => {
 let response = await fetch("/api/users")
 return response.json()
})

let usersSlice = createSlice({
 name: "users",
 initialState: { list: [], loading: false, error: null },
 reducers: {},
 extraReducers: (builder) => {
 builder
 .addCase(fetchUsers.pending, (state) => {
 state.loading = true
 })
 .addCase(fetchUsers.fulfilled, (state, action) => {
 state.loading = false
 state.list = action.payload
 })
 .addCase(fetchUsers.rejected, (state, action) => {
 state.loading = false
 state.error = action.error.message
 })
 }
})
```

JOTAI

Atomic state management:

npm install jotai

```
import { atom, useAtom, useAtomValue, useSetAtom } from "jotai"

let countAtom = atom(0)
let doubleCountAtom = atom((get) => get(countAtom) * 2)
```

```
function Counter() {
 let [count, setCount] = useAtom(countAtom)
 let doubleCount = useAtomValue(doubleCountAtom)

 return (
 <div>
 <p>Count: {count}</p>
 <p>Double: {doubleCount}</p>
 <button onClick={() => setCount(c => c + 1)}>+</
button>
 </div>
 )
}
```

Async atoms:

```
let userAtom = atom(async () => {
 let response = await fetch("/api/user")
 return response.json()
})

function User() {
 let user = useAtomValue(userAtom)
 return <p>{user.name}</p>
}
```

Writable derived atoms:

```
let celsiusAtom = atom(0)

let fahrenheitAtom = atom(
 (get) => get(celsiusAtom) * 9/5 + 32,
 (get, set, newValue) => {
 set(celsiusAtom, (newValue - 32) * 5/9)
 }
)
```

RECOIL

Facebook's state management:

npm install recoil

```
import { RecoilRoot, atom, selector, useRecoilState, useRecoilValue
} from "recoil"

let textState = atom({
 key: "textState",
 default: ""
})

let charCountState = selector({
 key: "charCountState",
 get: ({ get }) => {
 let text = get(textState)
 return text.length
 }
})

function TextInput() {
 let [text, setText] = useRecoilState(textState)

 return (
 <input value={text} onChange={(e) =>
setText(e.target.value)} />
 )
}

function CharacterCount() {
 let count = useRecoilValue(charCountState)
 return <p>Character count: {count}</p>
}

function App() {
 return (
```

```
  <RecoilRoot>
  <TextInput />
  <CharacterCount />
  </RecoilRoot>
 )
}
```

COMPARISON

Zustand:
- Simple API
- No boilerplate
- Flexible patterns
- Great for most apps

Redux Toolkit:
- Predictable
- Great DevTools
- Middleware support
- Best for large teams

Jotai:
- Atomic model
- Fine-grained updates
- Minimal re-renders
- Good for complex state

Recoil:
- Facebook's solution
- Atom-based
- Async selectors
- Concurrent mode ready

CHOOSING A LIBRARY

Start with React state + Context.

If that becomes complex, consider:
- Zustand for simplicity
- Redux Toolkit for large apps
- Jotai for fine-grained control

Most apps work fine with Zustand.

MIGRATION TIPS

From Context to Zustand:

```
let useAuth = create((set) => ({
 user: null,
 login: (user) => set({ user }),
 logout: () => set({ user: null })
}))
```

From Redux to Zustand:

```
let useStore = create((set, get) => ({
 todos: [],
 addTodo: (text) => set({
 todos: [...get().todos, { id: Date.now(), text, done: false }]
 }),
 toggleTodo: (id) => set({
 todos: get().todos.map(t =>
 t.id === id ? { ...t, done: !t.done } : t
 )
 })
}))
```

PRACTICE

Set up Zustand store.

Create Redux Toolkit slice.

Use Jotai atoms.

Compare performance.

Migrate from context.

WHAT YOU LEARNED

You learned Zustand basics.
You learned Redux Toolkit.
You learned Jotai.
You learned Recoil basics.
You learned when to use each.
You learned migration tips.

The next chapter teaches server components.
BOOK 17: REACT ADVANCED
Chapter 11: Server Components

WHAT ARE SERVER COMPONENTS

Server Components render on the server only.

Benefits:
- Direct database access
- Smaller bundle size

- Faster initial load
- No client-side JavaScript

Part of React 18+ with frameworks like Next.js.

SERVER VS CLIENT COMPONENTS

Server Components (default in Next.js App Router):
- Run on server only
- Cannot use hooks (useState, useEffect)
- Cannot use browser APIs
- Can access databases directly
- Can use async/await

Client Components (add "use client"):
- Run on client
- Can use hooks
- Can use browser APIs
- Have interactivity

BASIC SERVER COMPONENT

app/page.js:

```
async function HomePage() {
 let posts = await db.posts.findMany()

 return (
 <div>
 <h1>Blog Posts</h1>
 <ul>
 {posts.map(post => (
 <li key={post.id}>{post.title}</li>
 ))}
```

```
 </ul>
 </div>
 )
}

export default HomePage
```

No API endpoint needed.
Component fetches data directly.

CLIENT COMPONENT

app/components/Counter.js:

```
"use client"

import { useState } from "react"

function Counter() {
 let [count, setCount] = useState(0)

 return (
 <button onClick={() => setCount(count + 1)}>
 Count: {count}
 </button>
 )
}

export default Counter
```

"use client" directive marks as client component.

MIXING SERVER AND CLIENT

Server component with client children:

app/page.js:

```
import Counter from "./components/Counter"
import PostList from "./components/PostList"

async function HomePage() {
 let posts = await db.posts.findMany()

 return (
 <div>
 <h1>Dashboard</h1>
 <Counter />
 <PostList posts={posts} />
 </div>
 )
}
```

Counter is client (interactive).
PostList can be server (just displays data).

DATA FETCHING

Direct database access:

```
async function UserProfile({ userId }) {
 let user = await db.users.findUnique({
 where: { id: userId }
 })

 if (!user) {
 return <p>User not found</p>
 }

 return (
 <div>
```

```
 <h1>{user.name}</h1>
 <p>{user.email}</p>
 </div>
 )
}
```

API call from server:

```
async function ProductList() {
 let response = await fetch("https://api.example.com/products", {
 cache: "force-cache"
 })
 let products = await response.json()

 return (
 <ul>
 {products.map(product => (
 <li key={product.id}>{product.name}</li>
 ))}
 </ul>
 )
}
```

CACHING AND REVALIDATION

Static data (cached):

```
let response = await fetch("https://api.example.com/data", {
 cache: "force-cache"
})
```

Revalidate periodically:

```
let response = await fetch("https://api.example.com/data", {
 next: { revalidate: 3600 }
```

})

No cache (always fresh):

```
let response = await fetch("https://api.example.com/data", {
 cache: "no-store"
})
```

PASSING DATA TO CLIENT

Server fetches, client receives:

```
async function ProductPage({ productId }) {
 let product = await db.products.findUnique({
 where: { id: productId }
 })

 return (
 <div>
 <h1>{product.name}</h1>
 <p>{product.description}</p>
 <AddToCartButton product={product} />
 </div>
 )
}
```

app/components/AddToCartButton.js:

```
"use client"

function AddToCartButton({ product }) {
 function handleClick() {
 addToCart(product.id)
 }
```

```
 return <button onClick={handleClick}>Add to Cart</button>
}

export default AddToCartButton
```

Data serialized from server to client.

STREAMING

Progressive rendering:

```
import { Suspense } from "react"

async function Dashboard() {
 return (
 <div>
 <h1>Dashboard</h1>

 <Suspense fallback={<p>Loading stats...</p>}>
 <Stats />
 </Suspense>

 <Suspense fallback={<p>Loading chart...</p>}>
 <Chart />
 </Suspense>

 <Suspense fallback={<p>Loading table...</p>}>
 <DataTable />
 </Suspense>
 </div>
 )
}

async function Stats() {
 let stats = await fetchStats()
 return <div>{stats.total}</div>
}
```

Each section streams as it loads.

SERVER ACTIONS

Handle forms without API routes:

app/page.js:

```
async function createPost(formData) {
 "use server"

 let title = formData.get("title")
 let content = formData.get("content")

 await db.posts.create({
 data: { title, content }
 })
}

function NewPostForm() {
 return (
 <form action={createPost}>
 <input name="title" placeholder="Title" />
 <textarea name="content" placeholder="Content" />
 <button type="submit">Create Post</button>
 </form>
 )
}
```

With useFormState:

```
"use client"

import { useFormState } from "react-dom"
```

```
function ContactForm() {
 let [state, formAction] = useFormState(submitForm, null)

 return (
 <form action={formAction}>
 <input name="email" type="email" />
 <button type="submit">Submit</button>
 {state?.error && <p>{state.error}</p>}
 {state?.success && <p>Submitted!</p>}
 </form>
 )
}
```

PATTERNS

Fetch in parent, render in child:

```
async function BlogPage() {
 let posts = await db.posts.findMany()

 return <PostList posts={posts} />
}

function PostList({ posts }) {
 return (
 <ul>
 {posts.map(post => (
 <PostItem key={post.id} post={post} />
 ))}
 </ul>
 )
}
```

Parallel data fetching:

```
async function Dashboard() {
```

```
 let [users, posts, stats] = await Promise.all([
 db.users.findMany(),
 db.posts.findMany(),
 db.stats.get()
 ])

 return (
 <div>
 <UserList users={users} />
 <PostList posts={posts} />
 <StatsDisplay stats={stats} />
 </div>
 )
}
```

LIMITATIONS

Server Components cannot:
- Use useState, useEffect, useContext
- Use browser APIs (window, document)
- Add event handlers
- Use custom hooks with client logic

Use "use client" for interactive parts.

BEST PRACTICES

1. Default to Server Components
 Only use client when needed.

2. Keep client components small
 Extract interactive parts.

3. Fetch data in Server Components

Pass as props to client.

4. Use Suspense for streaming
Progressive loading.

5. Colocate data fetching
Fetch where data is used.

PRACTICE

Create server component with data.

Add client component for interactivity.

Use Suspense for streaming.

Implement server actions.

Mix server and client effectively.

WHAT YOU LEARNED

You learned server vs client components.
You learned data fetching patterns.
You learned caching strategies.
You learned streaming with Suspense.
You learned server actions.
You learned best practices.

The next chapter brings everything together with practice programs.

BOOK 17: REACT ADVANCED
Chapter 12: Practice React Programs

This chapter combines everything from the book.

Four complete programs using:
- useReducer for complex state
- Advanced context patterns
- Compound components
- Custom hooks
- Portals
- Server components
- Testing

PROGRAM 1: SHOPPING CART WITH USEREDUCER

Full cart system with complex state management.

The reducer handles all cart operations. Each case returns new state without mutating the original.

cartReducer.js:

```
let initialState = {
 items: [],
 discount: null,
 shipping: 0
}

function cartReducer(state, action) {
 switch (action.type) {
 case "ADD_ITEM": {
 let existing = state.items.find(
 item => item.id === action.payload.id
 )
```

```
if (existing) {
return {
...state,
items: state.items.map(item =>
item.id === action.payload.id
? { ...item, quantity: item.quantity + 1 }
: item
)
}
}

return {
...state,
items: [...state.items, { ...action.payload, quantity: 1 }]
}
}

case "REMOVE_ITEM": {
return {
...state,
items: state.items.filter(
item => item.id !== action.payload.id
)
}
}

case "UPDATE_QUANTITY": {
if (action.payload.quantity <= 0) {
return {
...state,
items: state.items.filter(
item => item.id !== action.payload.id
)
}
}

return {
...state,
items: state.items.map(item =>
```

```
item.id === action.payload.id
? { ...item, quantity: action.payload.quantity }
: item
)
}
}

case "APPLY_DISCOUNT": {
return {
...state,
discount: action.payload
}
}

case "SET_SHIPPING": {
return {
...state,
shipping: action.payload
}
}

case "CLEAR_CART": {
return initialState
}

default:
return state
}
}

export { cartReducer, initialState }
```

The context provides cart state and dispatch separately. This prevents unnecessary rerenders when only dispatch is needed.

CartContext.js:

```
"use client"
```

```
import { createContext, useContext, useReducer } from "react"
import { cartReducer, initialState } from "./cartReducer"

let CartContext = createContext(null)
let CartDispatchContext = createContext(null)

function CartProvider({ children }) {
 let [cart, dispatch] = useReducer(cartReducer, initialState)

 return (
 <CartContext.Provider value={cart}>
 <CartDispatchContext.Provider value={dispatch}>
 {children}
 </CartDispatchContext.Provider>
 </CartContext.Provider>
 )
}

function useCart() {
 let context = useContext(CartContext)
 if (!context) {
 throw new Error("useCart must be used within CartProvider")
 }
 return context
}

function useCartDispatch() {
 let context = useContext(CartDispatchContext)
 if (!context) {
 throw new Error("useCartDispatch must be used within
CartProvider")
 }
 return context
}

function useCartActions() {
 let dispatch = useCartDispatch()

 return {
 addItem: (product) => dispatch({
```

```
 type: "ADD_ITEM",
 payload: product
 }),
 removeItem: (id) => dispatch({
 type: "REMOVE_ITEM",
 payload: { id }
 }),
 updateQuantity: (id, quantity) => dispatch({
 type: "UPDATE_QUANTITY",
 payload: { id, quantity }
 }),
 applyDiscount: (code) => dispatch({
 type: "APPLY_DISCOUNT",
 payload: code
 }),
 setShipping: (cost) => dispatch({
 type: "SET_SHIPPING",
 payload: cost
 }),
 clearCart: () => dispatch({ type: "CLEAR_CART" })
 }
}

export { CartProvider, useCart, useCartDispatch, useCartActions }
```

Custom hook calculates totals with memoization. Only recalculates when cart changes.

useCartTotals.js:

```
import { useMemo } from "react"
import { useCart } from "./CartContext"

function useCartTotals() {
 let cart = useCart()

 let totals = useMemo(() => {
 let subtotal = cart.items.reduce((sum, item) => {
 return sum + (item.price * item.quantity)
```

```
 }, 0)

 let discountAmount = 0
 if (cart.discount) {
 if (cart.discount.type === "percentage") {
 discountAmount = subtotal * (cart.discount.value / 100)
 } else {
 discountAmount = cart.discount.value
 }
 }

 let total = subtotal - discountAmount + cart.shipping

 return {
 subtotal,
 discountAmount,
 shipping: cart.shipping,
 total,
 itemCount: cart.items.reduce((sum, item) => sum +
item.quantity, 0)
 }
 }, [cart])

 return totals
}

export { useCartTotals }
```

Individual cart item with quantity controls.

CartItem.js:

```
"use client"

import { useCartActions } from "./CartContext"

function CartItem({ item }) {
 let { updateQuantity, removeItem } = useCartActions()
```

```
 return (
 <div className="cart-item">
 <img src={item.image} alt={item.name} />

 <div className="item-details">
 <h3>{item.name}</h3>
 <p>${item.price.toFixed(2)}</p>
 </div>

 <div className="quantity-controls">
 <button onClick={() => updateQuantity(item.id, item.quantity -
1)}>
 -
 </button>
 <span>{item.quantity}</span>
 <button onClick={() => updateQuantity(item.id, item.quantity
+ 1)}>
 +
 </button>
 </div>

 <p className="item-total">
 ${(item.price * item.quantity).toFixed(2)}
 </p>

 <button onClick={() => removeItem(item.id)}
className="remove">
 Remove
 </button>
 </div>
 )
}

export default CartItem
```

Summary displays totals and handles discount codes.

CartSummary.js:

```
"use client"

import { useState } from "react"
import { useCartActions } from "./CartContext"
import { useCartTotals } from "./useCartTotals"

let discountCodes = {
 "SAVE10": { type: "percentage", value: 10 },
 "SAVE20": { type: "percentage", value: 20 },
 "FLAT50": { type: "fixed", value: 50 }
}

function CartSummary() {
 let [code, setCode] = useState("")
 let [error, setError] = useState("")
 let { applyDiscount } = useCartActions()
 let totals = useCartTotals()

 function handleApplyCode() {
 let discount = discountCodes[code.toUpperCase()]

 if (discount) {
 applyDiscount(discount)
 setError("")
 setCode("")
 } else {
 setError("Invalid discount code")
 }
 }

 return (
 <div className="cart-summary">
 <h2>Order Summary</h2>

 <div className="summary-line">
 <span>Subtotal ({totals.itemCount} items)</span>
 <span>${totals.subtotal.toFixed(2)}</span>
 </div>

 {totals.discountAmount > 0 && (
```

```
 <div className="summary-line discount">
 <span>Discount</span>
 <span>-${totals.discountAmount.toFixed(2)}</span>
 </div>
 )}

 <div className="summary-line">
 <span>Shipping</span>
 <span>
 {totals.shipping === 0 ? "Free" : "$" +
totals.shipping.toFixed(2)}
 </span>
 </div>

 <div className="summary-line total">
 <span>Total</span>
 <span>${totals.total.toFixed(2)}</span>
 </div>

 <div className="discount-input">
 <input
 value={code}
 onChange={e => setCode(e.target.value)}
 placeholder="Discount code"
 />
 <button onClick={handleApplyCode}>Apply</button>
 </div>

 {error && <p className="error">{error}</p>}

 <button className="checkout-button">
 Proceed to Checkout
 </button>
 </div>
 )
}

export default CartSummary
```

App wraps everything in provider.

App.js:

```
import { CartProvider } from "./CartContext"
import Cart from "./Cart"
import ProductList from "./ProductList"

function App() {
 return (
  <CartProvider>
  <div className="app">
  <ProductList />
  <Cart />
  </div>
  </CartProvider>
 )
}

export default App
```

PROGRAM 2: MODAL SYSTEM WITH PORTALS

Reusable modal using compound component pattern.

Modal uses context to share open/close state between its parts.

Modal.js:

```
"use client"

import { createContext, useContext, useState } from "react"
import { createPortal } from "react-dom"

let ModalContext = createContext(null)
```

```
function Modal({ children }) {
 let [isOpen, setIsOpen] = useState(false)

 let open = () => setIsOpen(true)
 let close = () => setIsOpen(false)

 return (
 <ModalContext.Provider value={{ isOpen, open, close }}>
 {children}
 </ModalContext.Provider>
 )
}

function useModal() {
 let context = useContext(ModalContext)
 if (!context) {
 throw new Error("Modal components must be used within Modal")
 }
 return context
}

Modal.Trigger = function ModalTrigger({ children, asChild }) {
 let { open } = useModal()

 if (asChild) {
 return children({ onClick: open })
 }

 return (
 <button onClick={open}>
 {children}
 </button>
 )
}

Modal.Content = function ModalContent({ children, title }) {
 let { isOpen, close } = useModal()

 if (!isOpen) return null
```

```
 return createPortal(
 <div className="modal-overlay" onClick={close}>
 <div
 className="modal-content"
 onClick={e => e.stopPropagation()}
 role="dialog"
 aria-modal="true"
 aria-labelledby="modal-title"
 >
 <div className="modal-header">
 <h2 id="modal-title">{title}</h2>
 <button onClick={close} aria-label="Close">
 X
 </button>
 </div>
 <div className="modal-body">
 {children}
 </div>
 </div>
 </div>,
 document.body
 )
}

Modal.Close = function ModalClose({ children }) {
 let { close } = useModal()

 return (
 <button onClick={close}>
 {children}
 </button>
 )
}

Modal.Footer = function ModalFooter({ children }) {
 return <div className="modal-footer">{children}</div>
}

export default Modal
```

Custom hook for standalone modal state.

useModalState.js:

```
import { useState, useCallback } from "react"

function useModalState(initialState = false) {
 let [isOpen, setIsOpen] = useState(initialState)

 let open = useCallback(() => setIsOpen(true), [])
 let close = useCallback(() => setIsOpen(false), [])
 let toggle = useCallback(() => setIsOpen(prev => !prev), [])

 return { isOpen, open, close, toggle }
}

export { useModalState }
```

Confirmation modal built on top of Modal.

ConfirmModal.js:

```
"use client"

import Modal from "./Modal"

function ConfirmModal({ trigger, title, message, onConfirm,
onCancel }) {
 return (
 <Modal>
 <Modal.Trigger>
 {trigger}
 </Modal.Trigger>

 <Modal.Content title={title}>
 <p>{message}</p>

 <Modal.Footer>
```

```
 <Modal.Close>Cancel</Modal.Close>
 <button
 onClick={onConfirm}
 className="danger"
 >
 Confirm
 </button>
 </Modal.Footer>
 </Modal.Content>
 </Modal>
 )
}

export default ConfirmModal
```

Usage example:

```
function DeleteButton({ itemId, onDelete }) {
 function handleConfirm() {
 onDelete(itemId)
 }

 return (
 <ConfirmModal
 trigger="Delete Item"
 title="Confirm Delete"
 message="Are you sure you want to delete this item?"
 onConfirm={handleConfirm}
 />
 )
}
```

Toast notifications also use portals. Provider manages toast list and auto-removal.

ToastContainer.js:

```
"use client"
```

```
import { createContext, useContext, useState, useCallback } from
"react"
import { createPortal } from "react-dom"

let ToastContext = createContext(null)

function ToastProvider({ children }) {
 let [toasts, setToasts] = useState([])

 let addToast = useCallback((message, type, duration) => {
 type = type || "info"
 duration = duration || 3000

 let id = Date.now()

 setToasts(prev => [...prev, { id, message, type }])

 setTimeout(() => {
 setToasts(prev => prev.filter(toast => toast.id !== id))
 }, duration)
 }, [])

 let removeToast = useCallback((id) => {
 setToasts(prev => prev.filter(toast => toast.id !== id))
 }, [])

 return (
 <ToastContext.Provider value={{ addToast }}>
 {children}
 {createPortal(
 <div className="toast-container">
 {toasts.map(toast => (
 <div
 key={toast.id}
 className={"toast toast-" + toast.type}
 >
 <span>{toast.message}</span>
 <button onClick={() => removeToast(toast.id)}>
 X
```

```
 </button>
 </div>
 ))}
 </div>,
 document.body
 )}
 </ToastContext.Provider>
 )
}

function useToast() {
 let context = useContext(ToastContext)
 if (!context) {
 throw new Error("useToast must be used within ToastProvider")
 }
 return context
}

export { ToastProvider, useToast }
```

Usage example:

```
function SaveButton() {
 let { addToast } = useToast()

 async function handleSave() {
 try {
 await saveData()
 addToast("Saved successfully!", "success")
 } catch (error) {
 addToast("Failed to save", "error")
 }
 }

 return <button onClick={handleSave}>Save</button>
}
```

PROGRAM 3: DATA TABLE WITH RENDER PROPS AND HOC

Flexible data table combining multiple patterns.

Base table handles sorting and filtering. Render prop allows custom row rendering.

DataTable.js:

```
"use client"

import { useState, useMemo } from "react"

function DataTable({
 data,
 columns,
 renderRow,
 renderEmpty,
 sortable,
 filterable
}) {
 sortable = sortable !== false
 filterable = filterable !== false

 let [sortConfig, setSortConfig] = useState({ key: null, direction:
"asc" })
 let [filterText, setFilterText] = useState("")

 let filteredData = useMemo(() => {
 if (!filterText) return data

 return data.filter(item => {
 return columns.some(column => {
 let value = item[column.key]
 return String(value)
 .toLowerCase()
 .includes(filterText.toLowerCase())
 })
```

```
})
}, [data, filterText, columns])

let sortedData = useMemo(() => {
if (!sortConfig.key) return filteredData

return [...filteredData].sort((a, b) => {
let aValue = a[sortConfig.key]
let bValue = b[sortConfig.key]

if (aValue < bValue) {
return sortConfig.direction === "asc" ? -1 : 1
}
if (aValue > bValue) {
return sortConfig.direction === "asc" ? 1 : -1
}
return 0
})
}, [filteredData, sortConfig])

function handleSort(key) {
setSortConfig(prev => ({
key,
direction: prev.key === key && prev.direction === "asc"
? "desc"
: "asc"
}))
}

return (
<div className="data-table">
{filterable && (
<input
value={filterText}
onChange={e => setFilterText(e.target.value)}
placeholder="Search..."
className="table-filter"
/>
)}
```

```
<table>
<thead>
<tr>
{columns.map(column => (
<th
key={column.key}
onClick={() => sortable && handleSort(column.key)}
className={sortable ? "sortable" : ""}
>
{column.label}
{sortConfig.key === column.key && (
<span>
{sortConfig.direction === "asc" ? " up" : " down"}
</span>
)}
</th>
))}
</tr>
</thead>
<tbody>
{sortedData.length === 0 ? (
<tr>
<td colSpan={columns.length}>
{renderEmpty ? renderEmpty() : "No data"}
</td>
</tr>
) : (
sortedData.map((item, index) =>
renderRow(item, index, columns)
)
)}
</tbody>
</table>

<p className="table-info">
Showing {sortedData.length} of {data.length} items
</p>
</div>
)
}
```

export default DataTable

Higher order component adds pagination to any table component.

withPagination.js:

"use client"

import { useState } from "react"

function withPagination(WrappedComponent, defaultPageSize) {
 defaultPageSize = defaultPageSize || 10

 return function PaginatedComponent(props) {
 let [currentPage, setCurrentPage] = useState(1)
 let [pageSize, setPageSize] = useState(defaultPageSize)

 let totalPages = Math.ceil(props.data.length / pageSize)
 let startIndex = (currentPage - 1) * pageSize
 let paginatedData = props.data.slice(startIndex, startIndex +
pageSize)

 function goToPage(page) {
 setCurrentPage(Math.max(1, Math.min(page, totalPages)))
 }

 return (

 <WrappedComponent
 {...props}
 data={paginatedData}
 />

 <button
 onClick={() => goToPage(1)}
 disabled={currentPage === 1}
 >

```
First
</button>
<button
onClick={() => goToPage(currentPage - 1)}
disabled={currentPage === 1}
>
Previous
</button>

<span>
Page {currentPage} of {totalPages}
</span>

<button
onClick={() => goToPage(currentPage + 1)}
disabled={currentPage === totalPages}
>
Next
</button>
<button
onClick={() => goToPage(totalPages)}
disabled={currentPage === totalPages}
>
Last
</button>

<select
value={pageSize}
onChange={e => {
setPageSize(Number(e.target.value))
setCurrentPage(1)
}}
>
<option value={5}>5 per page</option>
<option value={10}>10 per page</option>
<option value={25}>25 per page</option>
<option value={50}>50 per page</option>
</select>
</div>
</div>
```

```
 )
 }
}

export default withPagination


Combining HOC with DataTable:

import DataTable from "./DataTable"
import withPagination from "./withPagination"

let PaginatedDataTable = withPagination(DataTable, 10)

let columns = [
 { key: "id", label: "ID" },
 { key: "name", label: "Name" },
 { key: "email", label: "Email" },
 { key: "status", label: "Status" }
]

function UserTable({ users }) {
 return (
 <PaginatedDataTable
 data={users}
 columns={columns}
 renderRow={(user, index, cols) => (
 <tr key={user.id}>
 <td>{user.id}</td>
 <td>{user.name}</td>
 <td>{user.email}</td>
 <td>
 <span className={"status-" + user.status}>
 {user.status}
 </span>
 </td>
 </tr>
 )}
 renderEmpty={() => (
 <p>No users found. Try adjusting your search.</p>
```

```
)}
/>
)
}
```

PROGRAM 4: DASHBOARD WITH SERVER COMPONENTS

Full dashboard using server and client components together.

Main page is server component. Uses Suspense for progressive loading.

app/dashboard/page.js:

```
import { Suspense } from "react"
import StatsCards from "./StatsCards"
import RecentOrders from "./RecentOrders"
import SalesChart from "./SalesChart"
import TopProducts from "./TopProducts"
import DashboardFilters from "./DashboardFilters"

async function DashboardPage() {
 return (
 <div className="dashboard">
 <header className="dashboard-header">
 <h1>Dashboard</h1>
 <DashboardFilters />
 </header>

 <Suspense fallback={<StatsCardsSkeleton />}>
 <StatsCards />
 </Suspense>

 <div className="dashboard-grid">
 <Suspense fallback={<ChartSkeleton />}>
 <SalesChart />
```

```
 </Suspense>

 <Suspense fallback={<TableSkeleton />}>
 <RecentOrders />
 </Suspense>

 <Suspense fallback={<ListSkeleton />}>
 <TopProducts />
 </Suspense>
 </div>
 </div>
 )
}

function StatsCardsSkeleton() {
 return (
 <div className="stats-cards skeleton">
 {[1, 2, 3, 4].map(i => (
 <div key={i} className="stat-card loading" />
 ))}
 </div>
 )
}

function ChartSkeleton() {
 return <div className="chart-container skeleton loading" />
}

function TableSkeleton() {
 return <div className="table-container skeleton loading" />
}

function ListSkeleton() {
 return <div className="list-container skeleton loading" />
}

export default DashboardPage
```

Stats component fetches directly. No API endpoint needed.

app/dashboard/StatsCards.js:

```
async function StatsCards() {
 let stats = await fetchStats()

 return (
 <div className="stats-cards">
 <StatCard
 title="Total Revenue"
 value={"$" + stats.revenue.toLocaleString()}
 change={stats.revenueChange}
 icon="dollar"
 />
 <StatCard
 title="Orders"
 value={stats.orders.toLocaleString()}
 change={stats.ordersChange}
 icon="shopping-cart"
 />
 <StatCard
 title="Customers"
 value={stats.customers.toLocaleString()}
 change={stats.customersChange}
 icon="users"
 />
 <StatCard
 title="Conversion Rate"
 value={stats.conversionRate + "%"}
 change={stats.conversionChange}
 icon="trending-up"
 />
 </div>
 )
}

function StatCard({ title, value, change, icon }) {
 let isPositive = change >= 0

 return (
```

```
 <div className="stat-card">
 <div className="stat-icon">{icon}</div>
 <div className="stat-content">
 <p className="stat-title">{title}</p>
 <p className="stat-value">{value}</p>
 <p className={"stat-change " + (isPositive ? "positive" :
"negative")}>
 {isPositive ? "+" : ""}{change}% from last month
 </p>
 </div>
 </div>
 )
}

async function fetchStats() {
 let response = await fetch("https://api.example.com/stats", {
 next: { revalidate: 60 }
 })
 return response.json()
}

export default StatsCards
```

Filters are client component for interactivity.

app/dashboard/DashboardFilters.js:

```
"use client"

import { useState } from "react"
import { useRouter, useSearchParams } from "next/navigation"

function DashboardFilters() {
 let router = useRouter()
 let searchParams = useSearchParams()
 let [dateRange, setDateRange] = useState(
 searchParams.get("range") || "30d"
 )
```

```
 function handleDateChange(newRange) {
 setDateRange(newRange)

 let params = new URLSearchParams(searchParams)
 params.set("range", newRange)
 router.push("/dashboard?" + params.toString())
 }

 return (
 <div className="dashboard-filters">
 <select
 value={dateRange}
 onChange={e => handleDateChange(e.target.value)}
 >
 <option value="7d">Last 7 days</option>
 <option value="30d">Last 30 days</option>
 <option value="90d">Last 90 days</option>
 <option value="1y">Last year</option>
 </select>

 <button onClick={() => router.refresh()}>
 Refresh Data
 </button>
 </div>
 )
}

export default DashboardFilters
```

Orders table is server component. Actions are client.

app/dashboard/RecentOrders.js:

```
import OrderActions from "./OrderActions"

async function RecentOrders() {
 let orders = await fetchRecentOrders()

 return (
```

```
 <div className="recent-orders">
 <h2>Recent Orders</h2>

 <table>
 <thead>
 <tr>
 <th>Order ID</th>
 <th>Customer</th>
 <th>Amount</th>
 <th>Status</th>
 <th>Actions</th>
 </tr>
 </thead>
 <tbody>
 {orders.map(order => (
 <tr key={order.id}>
 <td>#{order.id}</td>
 <td>{order.customer}</td>
 <td>${order.amount.toFixed(2)}</td>
 <td>
 <span className={"status-" + order.status}>
 {order.status}
 </span>
 </td>
 <td>
 <OrderActions orderId={order.id} />
 </td>
 </tr>
 ))}
 </tbody>
 </table>
 </div>
 )
}

async function fetchRecentOrders() {
 let response = await fetch("https://api.example.com/orders/
recent", {
 cache: "no-store"
 })
```

```
 return response.json()
}

export default RecentOrders
```

Order actions handle mutations with transitions.

app/dashboard/OrderActions.js:

```
"use client"

import { useTransition } from "react"
import { updateOrderStatus } from "./actions"

function OrderActions({ orderId }) {
 let [isPending, startTransition] = useTransition()

 function handleStatusChange(newStatus) {
 startTransition(async () => {
 await updateOrderStatus(orderId, newStatus)
 })
 }

 return (
 <div className="order-actions">
 <select
 onChange={e => handleStatusChange(e.target.value)}
 disabled={isPending}
 >
 <option value="">Change status</option>
 <option value="processing">Processing</option>
 <option value="shipped">Shipped</option>
 <option value="delivered">Delivered</option>
 <option value="cancelled">Cancelled</option>
 </select>

 {isPending && <span className="loading-spinner" />}
 </div>
 )
```

```
}

export default OrderActions
```

Server action handles the mutation and revalidates.

app/dashboard/actions.js:

```
"use server"

import { revalidatePath } from "next/cache"

async function updateOrderStatus(orderId, status) {
 await fetch("https://api.example.com/orders/" + orderId, {
 method: "PATCH",
 headers: { "Content-Type": "application/json" },
 body: JSON.stringify({ status })
 })

 revalidatePath("/dashboard")
}

export { updateOrderStatus }
```

TESTING THE PROGRAMS

Tests for cart context:

CartContext.test.js:

```
import { renderHook, act } from "@testing-library/react"
import { CartProvider, useCart, useCartActions } from "./
CartContext"

function wrapper({ children }) {
 return <CartProvider>{children}</CartProvider>
```

```
}

test("adds item to cart", () => {
 let { result } = renderHook(
 () => ({ cart: useCart(), actions: useCartActions() }),
 { wrapper }
 )

 act(() => {
 result.current.actions.addItem({
 id: 1,
 name: "Test Product",
 price: 10
 })
 })

 expect(result.current.cart.items).toHaveLength(1)
 expect(result.current.cart.items[0].quantity).toBe(1)
})

test("increases quantity for existing item", () => {
 let { result } = renderHook(
 () => ({ cart: useCart(), actions: useCartActions() }),
 { wrapper }
 )

 act(() => {
 result.current.actions.addItem({ id: 1, name: "Test", price: 10 })
 result.current.actions.addItem({ id: 1, name: "Test", price: 10 })
 })

 expect(result.current.cart.items).toHaveLength(1)
 expect(result.current.cart.items[0].quantity).toBe(2)
})

test("removes item from cart", () => {
 let { result } = renderHook(
 () => ({ cart: useCart(), actions: useCartActions() }),
 { wrapper }
 )
```

```
 act(() => {
 result.current.actions.addItem({ id: 1, name: "Test", price: 10 })
 result.current.actions.removeItem(1)
 })

 expect(result.current.cart.items).toHaveLength(0)
})

test("applies discount", () => {
 let { result } = renderHook(
 () => ({ cart: useCart(), actions: useCartActions() }),
 { wrapper }
 )

 act(() => {
 result.current.actions.applyDiscount({
 type: "percentage",
 value: 10
 })
 })

 expect(result.current.cart.discount.value).toBe(10)
})
```

Tests for modal:

Modal.test.js:

```
import { render, screen, fireEvent } from "@testing-library/react"
import Modal from "./Modal"

test("opens modal on trigger click", () => {
 render(
 <Modal>
 <Modal.Trigger>Open</Modal.Trigger>
 <Modal.Content title="Test Modal">
 <p>Modal content</p>
 </Modal.Content>
```

```
 </Modal>
 )

 expect(screen.queryByText("Modal
content")).not.toBeInTheDocument()

 fireEvent.click(screen.getByText("Open"))

 expect(screen.getByText("Modal content")).toBeInTheDocument()
})

test("closes modal on close button click", () => {
 render(
 <Modal>
 <Modal.Trigger>Open</Modal.Trigger>
 <Modal.Content title="Test Modal">
 <p>Modal content</p>
 <Modal.Close>Close</Modal.Close>
 </Modal.Content>
 </Modal>
 )

 fireEvent.click(screen.getByText("Open"))
 expect(screen.getByText("Modal content")).toBeInTheDocument()

 fireEvent.click(screen.getByText("Close"))
 expect(screen.queryByText("Modal
content")).not.toBeInTheDocument()
})

test("closes modal on overlay click", () => {
 render(
 <Modal>
 <Modal.Trigger>Open</Modal.Trigger>
 <Modal.Content title="Test Modal">
 <p>Modal content</p>
 </Modal.Content>
 </Modal>
 )
```

```
 fireEvent.click(screen.getByText("Open"))

 fireEvent.click(document.querySelector(".modal-overlay"))

 expect(screen.queryByText("Modal
content")).not.toBeInTheDocument()
})
```

WHAT YOU BUILT

Program 1: Shopping cart
- useReducer for complex state
- Context for global access
- Custom hooks for actions
- Memoized calculations

Program 2: Modal system
- Compound component pattern
- Portal rendering
- Toast notifications
- Flexible API

Program 3: Data table
- Render props for rows
- HOC for pagination
- Sorting and filtering
- Reusable patterns

Program 4: Dashboard
- Server components for data
- Client components for interactivity
- Suspense for streaming
- Server actions for mutations

PRACTICE

Build your own cart with different features.

Create a dropdown using compound components.

Make a data grid with virtual scrolling.

Build a dashboard with real API.

Write tests for your components.

WHAT YOU LEARNED

You applied useReducer for complex state.

You built compound components.

You used portals for modals and toasts.

You combined render props and HOCs.

You built server component architecture.

You wrote tests for all patterns.

BOOK COMPLETE

You now understand advanced React patterns.

You can build complex state with useReducer.

You can create flexible component APIs.

You can use portals for overlays.

You can optimize with proper patterns.

You can leverage server components.

You can test React applications.

These patterns are used in production applications worldwide.

Continue building and exploring React.